Tesis de la Carrera de
Doctorado en Ingeniería Nuclear

Transporte de neutrones en la nube

[redacted]
Doctorando

[redacted]
Co-director

[redacted]
Co-director

Miembros del Jurado

[redacted]

Junio 2024

[redacted]

[redacted]
Argentina

A mis increíbles hijos Tomás y Máximo
y a mi hermosa esposa Natalia,
que me han acompañado en esta montaña rusa.

Y a todos los que entrenan laterales con sandías.

Resumen

Transporte de neutrones en la nube

Las herramientas neutrónicas a nivel de núcleo tradicionales suelen resolver la ecuación de difusión de neutrones multigrupo sobre mallas estructuradas hexaédricas. Aunque este enfoque puede ser razonable para reactores de potencia de agua liviana, los núcleos en los cuales el moderador está separado del refrigerante—como por ejemplo los reactores de potencia de agua pesada y algunos reactores de investigación—no pueden ser representados en forma precisa con mallas estructuradas, especialmente si las barras de control están inclinadas. En este trabajo, mostramos cómo podemos usar una herramienta libre y abierta que permite escalabilidad en paralelo corriendo en la nube para resolver ecuaciones en derivadas parciales discretizadas espacialmente con el método de elementos finitos para resolver neutrónica a nivel de núcleo con el método angular de ordenadas discretas S_N multigrupo. Esta herramienta, llamada FeenoX y desarrollada desde cero usando la filosofía de programación Unix, puede resolver PDEs genéricas al proveer un mecanismo basado en puntos de entrada arbitrarios usando apuntadores a funciones de C que construyen los objetos elementales de la formulación FEM. También permite escalar en paralelo utilizando el estándar MPI de forma que pueda ser lanzada sobre varios servidores en la nube. De esta manera, en principio, problemas arbitrariamente grandes pueden ser divididos en trozos más pequeños con técnicas de descomposición de dominio para poder franquear las limitaciones usuales en términos de memoria RAM. Dos de las PDEs que esta versión inicial del código puede resolver incluyen difusión de neutrones multigrupo y transporte de neutrones usando el método de ordenadas discretas.

Esta tesis explica la matemática de la ecuación de transporte de neutrones, cómo la aproximación de difusión puede ser derivada a partir de la primera y dos de las posibles discretizaciones en espacio y ángulo de ambas ecuaciones. También discute el diseño e implementación de FeenoX, que cumple un conjunto de especificaciones de requerimientos (SRS) ficticio—pero plausible—proponiendo un documento de diseño (SDS) explicando cómo la herramienta desarrollada aborda cada uno de los requerimientos del pliego. En el capítulo de resultados se resuelven diez problemas neutrónicos. Todos ellos necesitan al menos una de las características distintivas de FeenoX: 1. simulación programática (derivada de la filosofía Unix); 2. mallas no estructuradas; 3. ordenadas discretas; 4. paralelización con MPI. Este trabajo sienta las bases para eventuales estudios numéricos avanzados comparando los esquemas de S_N y difusión para análisis de reactores a nivel de núcleo.

Palabras clave transporte de neutrones, difusión de neutrones, computación en la nube, computación de alto rendimiento, elementos finitos, mallas no estructuradas

Revisión 2024-07-01–f53dd0b

Abstract

Neutron transport in the cloud

Traditional core-level neutronic computational tools are focused on solving the multigroup neutron diffusion equation over structured hexahedral grids. While this approach might be reasonable for light-water power reactors, cores where the moderator is separated from the coolant—such as heavy-water power plants and some research reactors—cannot be represented accurately with structured grids, especially if the control rods are slanted. In this work, we show how a free and open-source cloud-first large-scale parallel computational tool aimed at solving partial differential equations spatially discretized using the finite element method can be used to solve core-level neutronics using the multigroup discrete ordinates S_N angular scheme. This tool, named FeenoX and developed from scratch using the Unix programming philosophy, can solve generic PDEs by providing a mechanism based on arbitrary entry points using C function pointers which build the elemental objects for the FEM formulation. It also allows to scale in parallel using the MPI standard in a way which is suitable to be launched on several cloud servers. This way, in principle, large problems can be split into several hosts using domain decomposition techniques overcoming the usual RAM limitations. Two of the PDEs that the initial version of the code can solve include multigroup neutron diffusion and neutron transport using the discrete ordinates method.

This thesis explains the mathematics of the neutron transport equation, how the diffusion approximation can be derived from the former and two of the many possible numerical discretizations in angle and space for both equations. It also discusses the design and implementation of the tool FeenoX, that fulfills a fictitious (but plausible) set of requirement specifications (SRS) by proposing a design document (SDS) explaining how the developed tool addresses each of the tender requirements. In the results chapter, ten neutronic problems are solved. All of them need at least one of the unfair advantages that FeenoX's features configure: 1. programmatic simulation (that derives from the Unix philosophy); 2. unstructured grids; 3. discrete ordinates; 4. parallelization using MPI. This work sets a basis for further numerical studies comparing S_N and diffusion schemes for advanced core-level reactor analysis.

Keywords core-level neutron transport, neutron diffusion, cloud computing, high-performance computing, finite elements, unstructured grids

Revision 2024-07-01–f53dd0b

Tabla de contenidos

1. Introducción

> En su programa de televisión junto a Antonio Carrizo, el personaje de "El Contra" le pregunta a un invitado DT:
> — Supongamos que van 45' del segundo tiempo y su equipo está atacando. ¿Qué prefiere? ¿Que le den un córner o un lateral?
> — ¿Qué pregunta es esa? Un córner, porque tengo una chance de llegar al área.
> — Con el lateral también. Nosotros allá en Villa Dálmine, durante la semana entrenamos laterales con sandías. Cuando llega el domingo, (lo mira a Antonio Carrizo) ¿sabés hasta dónde tiramos la pelota?
> *Juan Carlos Calabró, 1993*

> Lo que yo daba por obvio dejó de serlo de un día para otro, concretando un choque que —debo admitir— ha vuelto a sucederme muchas veces desde entonces. Si hoy no me detengo a pensar un momento, vivo convencido de que todo el mundo es ingeniero nuclear o doctor en física.
> [redacted] *La Singularidad, 2006*

1. En los congresos académicos sobre métodos numéricos y aplicaciones, muchas veces uno puede ver un esquema similar al siguiente:

    a. Se muestra el estado del arte y se describen las falencias de un cierto algoritmo tradicional.
    b. Se propone una nueva idea o metodología para mejorar dicho estado del arte.
    c. Se aplican las novedades a un caso sencillo donde, efectivamente, la eficiencia o la precisión mejoran sensiblemente.
    d. Se indica que los próximos pasos serán resolver no ya problemas canónicos sino problemas de interés industrial.

    Pero también muchas veces, el nuevo desarrollo nunca llega efectivamente a la industria. El ímpetu de la investigación se va difundiendo a lo ancho de nuevas asignaciones de fondos de investigación que se enfocan en otras áreas, de gente que cambia de ideas, se jubila, etc. Las aplicaciones quedan en una lista de tareas "por hacer" (TO-DOs).

2. Los proyectos de ingeniería industriales están indefectiblemente restringidos por los tres vértices del triángulo de gerenciamiento de proyectos costos-alcance-calidad. Cualquier tarea de tipo de I+D+i en temas de modelado numérico debe necesariamente

   a. Ser pagada por el cliente.
   b. Ser aprobada por el cliente (o por quien el cliente designe, incluyendo agencias de regulación).
   c. Ser gerenciada de forma tal que los costos sean menores a los precios de venta.

   Durante el desarrollo de este tipo de proyectos se suelen identificar muchas ideas y tareas que podrían llevarse a cabo para generar nuevas oportunidades técnicas y comerciales. Pero la mayoría de las veces vienen nuevos proyectos con clientes diferentes y necesidades ortogonales. Las ideas quedan en una lista de TO-DOs.

3. En el ambiente de emprendedurismo, hay start ups relacionadas a temas de modelado numéricos de fenómenos complejos con interés en ingeniería. A menos que los fundadores tengan habilidades excepcionales como las de Henry Ford que en lugar de desarrollar "caballos más rápidos" creó una industria global desde cero, los casos de éxito están relacionados a tecnologías no muy disruptivas en sí mismas que termina en una adquisición por parte de una gran compañía existente, con sus inercias e idiosincrasias corporativas. Más aún, en el ámbito nuclear las regulaciones y la financiación de nuevas instalaciones es tan complicada que necesariamente deben involucrarse entidades oficiales que complican mucho más aún la dinámica característica de las start-ups. El ímpetu se va difundiendo en organigramas rígidos poco amenos a la innovación. La mayoría de las ideas quedan en una lista de TO-DOs.

Habiendo explorado profesionalmente con bastante detalle cada uno de estos tres puntos durante los últimos quince años, he decido escribir esta tesis de doctorado en forma atípica y excepcional—en el sentido de "excepción". Empezando por mi edad al momento de entregar y defender esta tesis y siguiendo por la forma de encarar los trabajos (como por ejemplo el propuesto en la presentación [68]). Los desarrollos aquí descriptos han sido claramente inspirados por estos tres puntos, pero fueron realizados en forma completamente independiente de estos tres ámbitos. Es esta una tesis escrita durante los fines de semana por un profesional de la industria del software de cálculo numérico sin los condicionamientos particulares de la academia, la industria ni el emprendedurismo. Representa entonces, una manera de poder evaluar—dentro de las capacidades y responsabilidades que me atañen—la diferencia entre lo urgente y lo importante como propone Mafalda (figura 1.1).

Figura 1.1.: Mafalda, lo urgente y lo importante.

El objetivo principal es entonces sentar las bases para que las tareas identificadas como potenciales generadoras de nuevas oportunidades técnicas y comerciales, mencionadas explícitamente en la sección 6.1, no queden solamente en una lista de puntos "nice to have".

El contenido general de esta tesis es una mezcla de

i. física de neutrones y reactores a nivel de núcleo
ii. programación en computación de alto rendimiento

que son justamente las dos mitades del título y los temas profesionales en los que me desempeñé en la academia, en la industria y en el mundo del emprendedurismo. Este contenido está distribuido en seis capítulos, incluido este, ordenados según el orden de presentación "por qué", "cómo" y "qué":

a. ¿Por qué?

- Capítulo 1: Introducción

b. ¿Cómo?

- Capítulo 2: Transporte y difusión de neutrones
- Capítulo 3: Esquemas de discretización numérica

c. ¿Qué?

- Capítulo 4: Implementación computacional
- Capítulo 5: Resultados

Debo reconocer que la extensión de esta tesis es mayor que la que me hubiese gustado que tenga. Pero también es cierto que he necesitado escribir todo este texto (y ecuaciones) ya que de esta forma he podido entender y procesar toda la información necesaria para hacer aportes originales a mis cuarenta años. El primer capítulo (why) contiene apreciaciones un tanto subjetivas pero basadas en experiencia sólida. Los siguientes dos capítulos (how) contienen desarrollos matemáticos harto conocidos pero, como allí se menciona, funcionan como una amalgama uniforme de varias fuentes de forma tal de generar una base teórica del desarrollo en sí mismo, y de permitirme "digerir" el cuerpo teórico detrás de los métodos numéricos relacionados al transporte de neutrones a nivel de núcleo. Los últimos dos capítulos (what) contienen el núcleo de la contribución original.

1.1. Cien años de programación

Durante mi paso por la industria nuclear en el completamiento de la Central Nuclear Atucha II (el punto 2 de la página 2) he tenido la experiencia de emplear herramientas computacionales de cálculo neutrónico, termohidráulico y de control [77]. Por razones que no viene al caso analizar, aún en la década de 2010, mucho del software empleado había sido diseñado originalmente varias décadas antes cuando los paradigmas computacionales eran radicalmente diferentes. Por ejemplo, la tabla 1.1 muestra un punto central de este paradigma: el costo de la hora de CPU de las computadoras usualmente utilizadas para cálculos nucleares en 1965 [90].

Cuando el sistema operativo Unix fue introducido a principios de la década de 1970 [50], los diseñadores ya habían previsto un gran descenso en los costos del hardware y un corrimiento de costos de

Computer	Monthly Rental	Relative Speed	First Delivery
CDC 3800	$ 50,000	1	Jan 66
CDC 6600	$ 80,000	6	Sep 64
CDC 6800	$ 85,000	20	Jul 67
GE 635	$ 55,000	1	Nov 64
IBM 360/62	$ 58,000	1	Nov 65
IBM 360/70	$ 80,000	2	Nov 65
IBM 360/92	$ 142,000	20	Nov 66
PHILCO 213	$ 78,000	2	Sep 65
UNIVAC 1108	$ 45,000	2	Aug 65

(a) Relative speed is expressed with reference to IBM 7030. Data for computers expected to appear after 1965 was estimated.

Tabla 1.1.: Tabla "Las nuevas computadoras de alta velocidad de 1965" de la referencia [90]. Los costos están expresados en dólares americanos de 1965 y pueden variar en un factor de dos. Un dólar de 1965 equivale a USD 10 de 2024.

CPU a ingeniería. De hecho una de las 17 reglas de la filosofía de Unix en las que se basa largamente el diseño de la herramienta computacional objeto de esta tesis (Apéndice C) se denomina "Regla de Economía" que indica que el software debe ser diseñado pensando en que el tiempo de la persona que usa el programa es mucho más valioso que el del hardware que lo ejecuta. Hoy en día, alquilar una hora de CPU de una computadora razonablemente rápida y con suficiente memoria para realizar cálculos nucleares estándar sale menos de 0,15 USD. Un servidor con 16Gb de RAM disponible 100% del tiempo se puede conseguir por veinte dólares al mes.

La forma de diseñar software de cálculo hoy en día debe ser, entonces, radicalmente diferente a la usada media docena de décadas atrás. En el capítulo 4 discutimos en detalle todas estas particularidades, pero el ejemplo clásico se reduce a aquellos programas de cálculo (sobretodo termohidráulicos) en los que la salida está compuesta exactamente por todos y cada una de los resultados calculados, incluyendo distribuciones espaciales y temporales. Esta decisión de diseño tiene sentido cuando es muy caro tener que volver a realizar un cálculo porque un cierto valor requerido no forma parte de la salida. Pero esto implica que el ingeniero a cargo del cálculo debe buscar y filtrar solamente aquellos resultados necesarios en medio de un pajar de información a un costo horario miles de veces superior a tener que volver a realizar el cálculo pidiendo explícitamente el resultado requerido, y nada más.

El diseño del sistema operativo Unix (y del lenguaje de programación C, estrechamente relacionado) ha dado en el clavo en muchos aspectos técnicos que hacen que su tecnología esté vigente como nunca más de cincuenta años después de las decisiones de diseño. Tanto es así que prácticamente todos los servidores públicos de Internet funcionan sobre alguna variante de este diseño. Más aún, la arquitectura es tan sólida que aunque en las décadas de 1990 y 2000 hayan aparecido muchas herramientas de cálculo basadas en Windows (la invasión del "X for Windows" que, en mi humilde opinión, ha sido perjudicial para la salud de la comunidad de la mecánica computacional), se ha probado que es técnica y económicamente mucho más eficiente recurrir a un esquema de alquiler de recursos computacionales (la nube pública) en lugar de recurrir a comprar y mantener servidores propios (on premise). Desde el punto de vista económico, lo segundo implica costos de capital

mientras que lo primero son costos de operación. Desde el punto de vista técnico, no tiene ningún sentido comprar hardware cuyo nivel de utilización será menor al 100%. En terminología de start ups: "rent, don't buy".

En este sentido, la herramienta computacional de cálculo desarrollada desde cero en esta tesis para resolver ecuaciones diferenciales en derivadas parciales fue diseñada para ser ejecutada *nativamente* en la nube. Como explicamos en detalle en la sección 4.4.6, haciendo un paralelismo con la nomenclatura de interfaces web donde hay diseños mobile friendly y mobile first, decimos que la herramienta es cloud first y no solamente cloud friendly.

A lo largo de otro paso profesional por el punto 2 en la industria de la consultoría y del desarrollo de software, aprendí que un esquema usual de contratación entre un cliente y un proveedor de software consiste en que el primero prepara un documento titulado "Software Requirements Specification" listando justamente los requerimientos técnicos que el software a comprar debe cumplir como una especie de pliego técnico particular. Entonces los potenciales proveedores preparan un documento de "Software Design Specification" en el que explican técnicamente cómo planean cumplir con los requerimientos. Una especie de oferta técnica al pliego. Teniendo en cuenta estas consideraciones (más la experiencia de los tres puntos del comienzo del capítulo), he decidido entonces organizar el diseño de una nueva herramienta partiendo primero de un SRS ficticio (pero plausible), pidiendo lo que me gustaría que una herramienta computacional cloud first cumpla. A partir de estos requerimientos, empecé a estudiar la forma de cumplirlos, implementarlos y documentarlos en un SDS. Ambos documentos forman parte de los apéndices de esta tesis (apéndices A y B).

Además del requerimiento de que la herramienta desarrollada corra en la nube, se requiere también que el software desarrollado sea libre y abierto. Este punto es de especial importancia tanto en la academia como en la industria (por diferentes razones en cada caso) y sus implicaciones son usualmente ignoradas, especialmente en la industria nuclear. En la sección 4.4.1 explicamos las razones de dicha importancia.

Otra característica, explicada en detalle en el capítulo 4, es que la arquitectura del código fue diseñada en forma tal que sea extensible en el sentido de poder agregar nuevas formulaciones de ecuaciones a resolver en forma razonablemente sencilla sin necesidad de tener que escribir un nuevo solver para cada ecuación. La forma de implementar esta característica se basa en un esquema de apuntadores a función resueltos en tiempo de ejecución según el tipo de problema que se requiere resolver definido en el archivo de entrada.

Un requerimiento importante es que la herramienta sea escalable en paralelo para permitir resolver problemas relativamente grandes con discretizaciones relativamente finas. En forma abstracta, la idea de paralelización de un código de cálculo se suele asociar a la posibilidad de obtener resultados en forma más rápida que en el caso serie sin paralelizar ya que, en principio, al disponer de más unidades de procesamiento es posible realizar más operaciones de coma flotante por unidad de tiempo. Pero desde el punto de vista de esta tesis, el principal objetivo *no* es el tiempo de cálculo sino la cantidad de memoria necesaria para poder resolver un cierto problema, como explicamos a continuación. En forma particular, y sin entrar en detalles técnicos, existen esencialmente tres tipos de paralelismo [84]:

Sistemas de memoria compartida (OpenMP) múltiples unidades de procesamiento vinculadas a un único espacio de direcciones de memoria.

Sistemas distribuidos (MPI) múltiples unidades computacionales, cada una con sus unidades de procesamiento y memoria, conectadas entre sí a través de redes de alta velocidad.

Unidades de procesamiento gráfico (GPU) utilizadas como co-procesadores para resolver problemas numéricamente intensivos.

Tanto en el caso OpenMP como GPU, los threads que corren en paralelo comparten la memoria física de acceso aleatorio (RAM). Entonces mientras mayor sea el problema, más memoria se necesitará. Eventualmente, para algún cierto tamaño de problema crítico, se llegará a un límite de memoria que no se podrá franquear. Pero en el caso de MPI [14], como los procesos paralelos pueden estar en diferentes computadoras físicas (o no), la memoria total disponible se puede hacer arbitrariamente grande agregando nuevos hosts al sistema distribuido. En efecto, en el capítulo 5 mostramos que para un tamaño de problema fijo la memoria por proceso MPI disminuye monótonamente con la cantidad de procesos. Al combinar esta capacidad con el requerimiento de que la herramienta pueda correr en la nube, en principio se podrían resolver problemas de tamaño arbitrario si se pudieran alquilar suficientes instancias cloud.

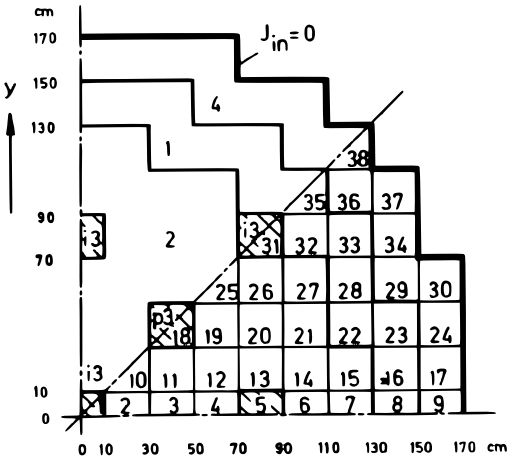
Observación. La biblioteca PETSc [7], que es la que usa la herramienta desarrollada en esta tesis para resolver los problemas ralos que resultan de discretizar ecuaciones diferenciales en derivadas parciales, basa su esquema de paralelización en el paradigma MPI. Sus desarrolladores manifiestan expresamente—tanto en forma escrita a través de correos electrónicos como en forma oral en las reuniones anuales de usuarios [42]—que no hay ninguna razón técnica para preferir el paradigma OpenMP sobre el MPI. Si bien hay esfuerzos para soportar OpenMP en PETSc, éstos apuntan solamente a dar soporte a código existente para que pueda aprovechar las ventajas de PETSc. Los desarrolladores recomiendan diseñar código nuevo basando en MPI por sobre OpenMP.

Observación. Con respecto a GPU, PETSc provee interfaces para los SDKs más comunes (CUDA, HIP, SYCL, Kokkos, etc.) que pueden descargar[1] operaciones de álgebra elemental en tiempo de ejecución con opciones de línea de comando. Una de las ventajas particulares de la filosofía Unix de hacer una sola cosa bien y re-utilizar las cosas que ya están bien hechas es que la herramienta desarrollada en esta tesis tiene soporte para GPU "gratis" aprovechando estas interfaces.

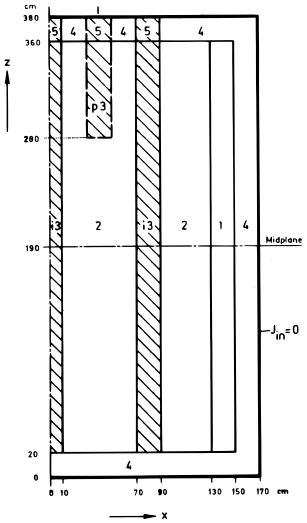
Combinando estos requerimientos del SRS (apéndice A) y la forma en la que se abordan desde el punto de vista del diseño en el SDS (apéndice B) e implementación (capítulo 4), considero que la contribución de esta tesis es original ya que no tengo conocimiento de la existencia de un software similar que cubra las mismas características requeridas. Tal vez existan algunas que cumplan un par de los requerimientos, o incluso alguna que cumpla una cierta fracción significativa. Pero ninguna el 100%. Esto es más notable aún teniendo en cuenta que el objeto principal de estudio de esta tesis es la neutrónica a nivel de núcleo, resuelta tanto con difusión como con ordenadas discretas sobre mallas no estructuradas. A modo de ejemplo de la clase de contribución que propongo, consideremos el Benchmark PWR 3D propuesto por la IAEA en 1976 [3]. La figura 1.2 muestra el resultado de haber resuelto el problema pero…

1. con una simetría 1/8 en lugar de la simetría 1/4 original,
2. con un reflector cilíndrico en lugar de un reflector compuesto por planos perpendiculares a los ejes cartesianos,

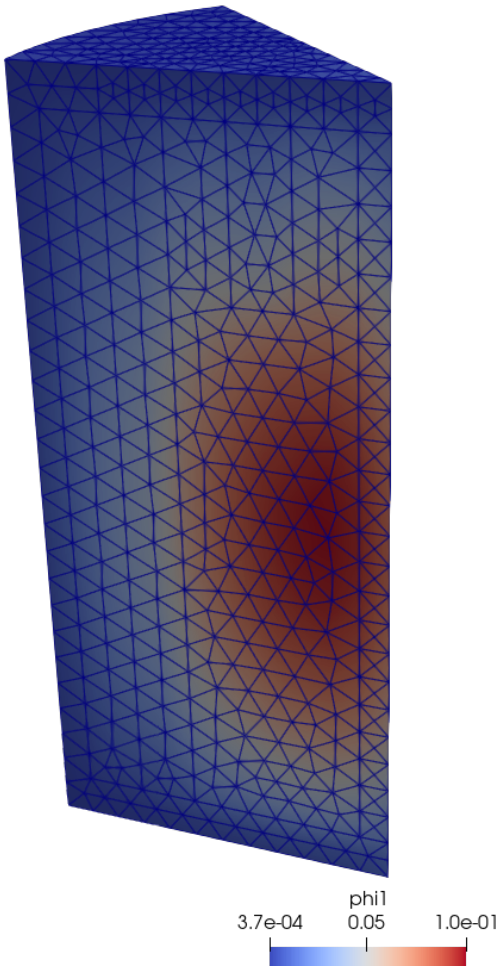
[1] En el sentido del inglés *offload*.



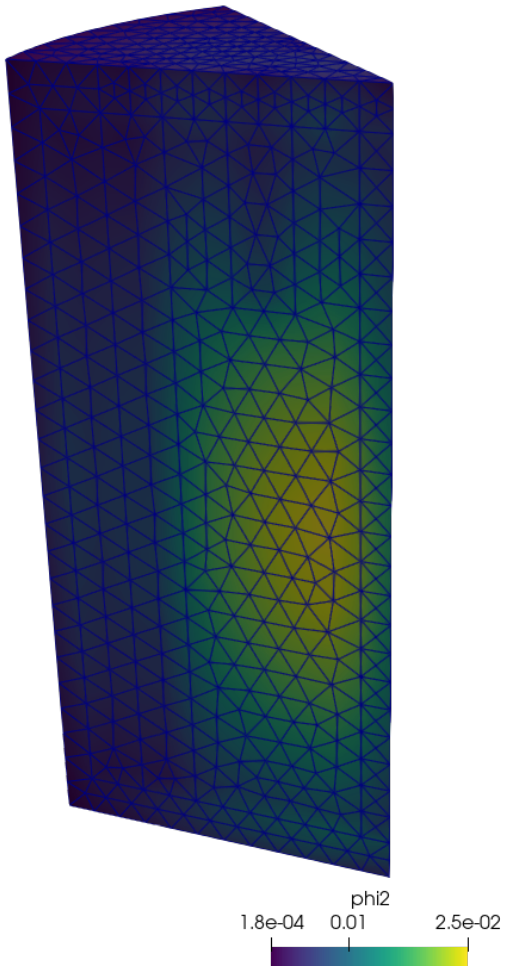
(a) Figura original vista superior



(b) Figura original vista lateral



(c) Flujo rápido ϕ_1



(d) Flujo térmico ϕ_2

Figura 1.2.: Benchmark PWR 3D de IAEA [3] para ilustrar las características distintivas de la formulación propuesta. Resolvemos en detalle este problema en la sección 5.3.

3. resuelto con una formulación S_4 de ordenadas discretas en lugar de la original de difusión, y
4. en paralelo utilizando cuatro procesos independientes.

Cada uno de estos cuatro puntos está detalladamente explicado en el cuerpo de la tesis y, junto con

i. la capacidad de extender el área de los problemas a resolver agregando nuevas formulaciones de ecuaciones discretizadas con el método de elementos finitos (ver el apéndice B para ejemplos por fuera de la neutrónica de núcleo)
ii. el diseño cloud first que permite realizar lo que se conoce como "simulación programática" sin necesidad de interactuar
iii. la discretización del dominio utilizando mallas no estructuradas, potencialmente realizando descomposición de dominio
iv. la posibilidad de escalar en paralelo mediante y poder resolver problemas de tamaño arbitrario

constituyen el unfair advantage—en el sentido del canvas de modelo de negocios—de la herramienta desarrollada [83].

Pero es la combinación de todos ellos la que configura la propuesta de la tesis, minuciosamente explicada en la sección 1.4: poder disponer de una herramienta computacional que permita resolver problemas de neutrónica a nivel de núcleo de tamaño arbitrario a través de la escalabilidad en paralelo basado en el estándar MPI. Lo importante es que, como mostramos en el capítulo 5, hacemos una descomposición del dominio y de "repartimos" la carga computacional—especialmente la memoria RAM que usualmente es el recurso limitante—en varios procesos. Esto hace posible resolver problemas formulados con S_N en mallas no estructuradas en principio de tamaño arbitrario y, eventualmente, poder compararlos con la aproximación de difusión que es computacionalmente mucho menos demandante. Para ello se necesita sobrepasar las limitaciones de las herramientas neutrónicas tradicionales (sección 1.3), que es la idea central de esta tesis.

Observación. Este trabajo sólo se enfoca en el desarrollo de la herramienta necesaria para realizar la comparación. Un estudio cuantitativo de la eficiencia de diferentes esquemas numéricos para hacer ingeniería neutrónica de núcleo implicaría un proyecto de ingeniería de varios hombre-años más sus costos indirectos asociados. En la sección 6.1 listamos algunos de los trabajos futuros que podría derivar de las bases sentadas en esta tesis.

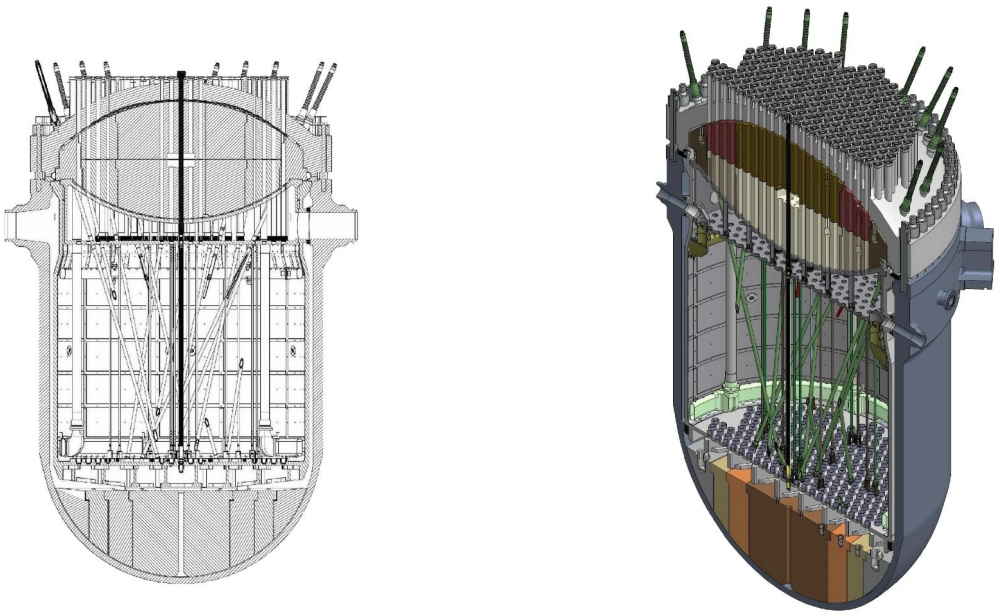
Finalmente, a modo personal debo notar que en el Proyecto Integrador de mi Carerra de Ingeniería Nuclear traté temas de control en loops de convección natural caóticos [60] y en la Tesis de Maestría en Ingeniería traté temas de inestabilidades termohidráulicas en presencia de una fuente de potencia de origen neutrónico [61]. Poder realizar una tesis de doctorado en temas de neutrónica de nivel de núcleo me permite cerrar en forma académica el lazo termohidráulica-neutrónica-control, que fue también el eje de mi participación profesional en el completamiento de la Central Nuclear Atucha II.

1.2. Historia de dos reactores

El reactor de la Central Nuclear Atucha I fue puesto en condición crítica el 13 de enero de 1974, constituyendo así la primera planta de generación nucleoeléctrica instalada en América Latina. Es

un reactor térmico de 357 MWe brutos, moderado por agua pesada con canales combustibles verticales refrigerados también por agua pesada a la misma presión que el moderador pero a diferente temperatura. Debido a que el combustible es uranio natural, la poca reactividad en exceso hace que el reactor deba tener un recambio de combustible continuo. Para ello existe una máquina de recambio que opera verticalmente sobre la tapa superior del reactor extrayendo elementos combustibles gastados e introduciendo frescos a un ritmo aproximado de uno por día. El resto del tiempo la máquina opera haciendo un mezclado[2] para homogeneizar el quemado de los combustibles y optimizar el quemado de extracción final. Dado que la parte superior del recipiente de presión debe quedar libre para que trabaje la máquina de recambio, los mecanismos de control de reactividad tanto primarios (las barras de control) como secundarios (sistema de inyección de boro de emergencia) deben entrar el núcleo en forma oblicua a los canales combustibles, configurando un diseño esencialmente único en el mundo.

Atucha II tiene un diseño similar aunque genera más del doble de potencia, 745 MWe brutos. Debido a una combinación de causas que quedan fuera del alcance de cualquier análisis, el contrato de construcción se firmó en 1980 pero la obra no se terminó hasta 2014. Detalles más detalles menos, el diseño del reactor es similar al de Atucha I (figura 1.3).



(a) Vista en corte

(b) Vista 3D

Figura 1.3.: Layout de canales y tubos guía de barras de control en el reactor de presión de la Central Nuclear Atucha II [77].

1.2.1. Neutrones contantes y sonantes

El martes 3 de junio de 2014 a las 9:03 el núcleo[3] de la Central Nuclear Atucha II logró mantener por primera vez una reacción nuclear de fisión en cadena autosostenida. Este suceso marcó un hi-

[2] En el sentido del inglés *shuffling*.
[3] En el sentido del inglés *core*.

to no sólo en la industria nuclear argentina sino también en mi carrera profesional. Durante cinco años y medio estuve trabajando desde TECNA S.A. junto a un equipo ingenieros de Nucleoeléctrica Argentina S.A. en el desarrollo de modelos y códigos de cálculo acoplados para predecir, estudiar y analizar el comportamiento de la central teniendo en cuenta neutrónica espacial, realimentaciones termohidráulicas y acciones del sistema de control, limitación y protección del reactor. Esa mañana pude presenciar de primera mano las indicaciones de los instrumentos que mostraban un incremento lineal en el tiempo de la señal de nivel de flujo neutrónico en el núcleo del reactor, que es lo que yo había leído en los libros de texto que debía suceder en un reactor crítico en presencia de una fuente independiente de neutrones. Para terminar de despejar cualquier clase de dudas (incluyendo físicas, tecnológicas e industriales), durante los siguientes meses continué dando soporte de ingeniería a las tareas de aumento escalonado de potencia hasta el cien por ciento, obteniendo evidencia experimental a prueba de escépticos de que realmente el reactor era capaz de generar potencia térmica a partir de la fisión del uranio mediante reacciones inducidas por neutrones.

1.2.2. Más de una rueda de auxilio

Aún cuando no son deseados, los imprevistos existen. Es por eso que todos nos aseguramos de que la rueda de auxilio de nuestro auto esté en condiciones antes de emprender un viaje más o menos largo ya que existe una probabilidad p_1 no nula de que se nos pinche una cubierta en el camino. ¿Pero por qué decimos *la* rueda de auxilio y no *las* ruedas de auxilio? ¿Acaso la probabilidad $p_2 \approx p_1^2$ de pinchar no una sino *dos* cubiertas en sucesos independientes no es también diferente de cero al fin y al cabo? Sí, claro, pero esa probabilidad $p_2 \ll 1$ es tan pequeña que no vale la pena el esfuerzo y el costo que implica llevar dos ruedas de auxilio en nuestro automóvil. Llegado el caso, llamamos a la grúa. En el diseño de centrales nucleares usamos un razonamiento similar: para todos los eventos cuyas probabilidades p_i de ocurrencia sean significativas (accidentes de base de diseño) debemos tomar precauciones; para el resto (accidentes fuera de la base de diseño), preparamos soluciones de contingencia.

En general, los reactores nucleares de potencia necesitan más de un único mecanismo de extinción de las reacciones de fisión. El primero consiste en las mismas barras de control, que son insertadas rápidamente dentro del núcleo para absorber neutrones y no permitir que las fisiones se auto-sostengan en el tiempo. Si bien la probabilidad de que este mecanismo falle es pequeña en términos absolutos, debemos incluir al menos un sistema más de extinción del reactor redundante, diverso e independiente. En el caso de las Centrales Nucleares tipo Atucha, el segundo sistema de extinción del reactor consiste en inyectar rápidamente una solución de ácido deuterobórico en el tanque del moderador. De esta forma, como los núcleos[4] de ^{10}Bo son grandes absorbentes de neutrones, las reacciones de fisión se extinguen a medida que el boro ingresa a la zona del núcleo del reactor.

1.2.3. La conexión europea

Fue condición necesaria para la puesta a crítico de la central, la preparación del Informe Final de Seguridad (FSAR) y su presentación a la Autoridad Regulatoria Nuclear. Si bien la mayor parte de la ingeniería necesaria para su elaboración fue realizada en el país, debido a ciertas características del

[4] En el sentido del inglés *nuclei*.

proyecto Atucha II—tanto técnicas como de gerenciamiento, con las que coincido completamente—para la evaluación de algunos aspectos relacionados al Capítulo 15 de Análisis de Accidentes fueron contratados consultores del exterior. En particular, el modelado de la actuación del sistema de inyección de boro de emergencia y su efecto sobre la neutrónica durante casos accidentales fue uno de estos aspectos. Por un lado, este estudio involucraba cierto know-how y técnicas que no estaban completamente desarrolladas en el país al momento de comenzar los trabajos de licenciamento de la central. Por otro lado, suele ser una buena práctica involucrar a grupos internacionales especializados, sobretodo en temas complejos, delicados y sensibles.

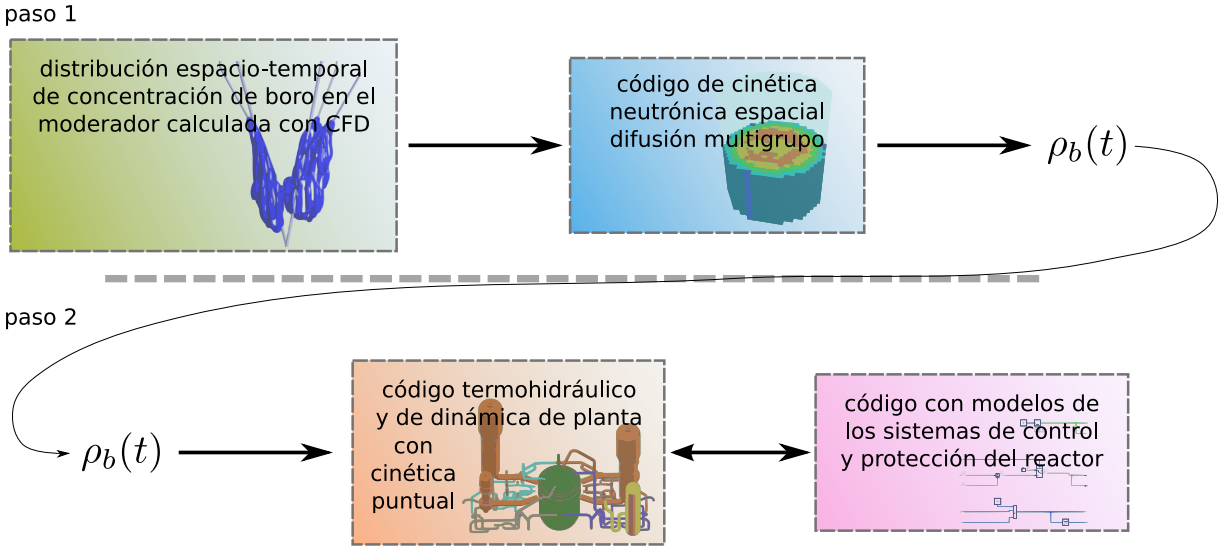
1.2.4. Aprendiendo de los que saben

A partir de la experiencia ganada desde la interacción con estos expertos y de las capacidades propias desarrolladas durante el proceso de licenciamento de Atucha II, que han servido incluso para que los consultores externos mejoren tanto sus propias capacidades como los resultados provistos, es que a comienzos de 2014 Nucleoeléctrica decidió que las tareas de actualización del Informe Final de Seguridad de la Central Nuclear Atucha I utilizando modelos, métodos y códigos según el estado del arte actual (i.e. los modelos, métodos y códigos usados para licenciar Atucha II) sean realizadas íntegramente en el país por ingenieros argentinos. En particular, la evaluación de la reactividad negativa insertada por el sistema de inyección de emergencia y de los efectos espaciales de la interacción neutrónica-termohidráulica durante accidentes recayó sobre nuestro equipo de trabajo, cuando anteriormente se habían contratado consultores europeos para realizar dichas tareas de ingeniería [76]. Durante 2014 hemos trabajado en el diseño de un esquema de cálculo acoplado basado en recursos de memoria compartida para permitir que durante el 2015 se realicen los estudios necesarios para analizar una treintena de accidentes de base de diseño que componen el Capítulo 15 del FSAR de la Central Nuclear Atucha I figura 1.4.

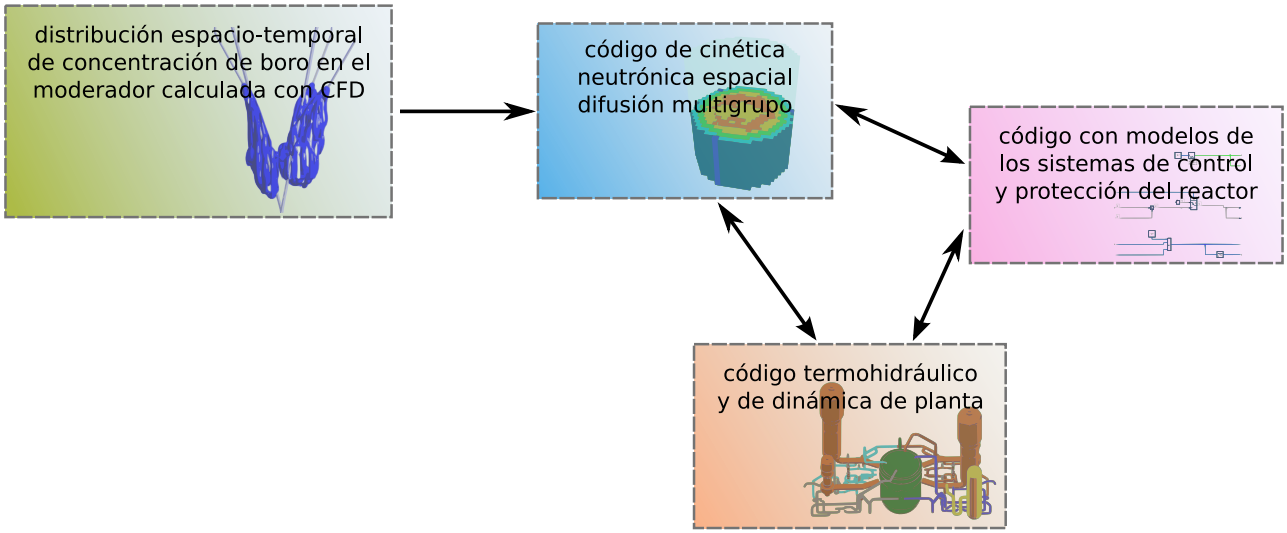
1.2.5. Claro como el agua pesada

En los reactores de agua pesada presurizada, si bien las funciones de refrigeración del combustible y moderación de los neutrones son realizadas por el mismo material (justamente agua pesada), las condiciones de temperatura a las que se encuentran refrigerante y moderador son diferentes. Incluso en ciertos diseños y/o condiciones operacionales, la presión puede ser diferente. Por lo tanto, desde el punto de vista neutrónico se deben considerar como materiales diferentes. El núcleo consiste en un arreglo periódico de canales refrigerantes con sus ejes paralelos entre sí. En los reactores tipo Atucha los canales se encuentran en forma vertical y con un arreglo sobre el plano transversal basado en triángulos equiláteros, mientras que éstos son horizontales y distribuidos como vértices de cuadrados en reactores tipo CANDU (figura 1.5). Los canales están inmersos en un gran tanque que contiene el moderador líquido, que usualmente se mantiene más frío que el refrigerante con el objetivo de mejorar la moderación y aumentar así el factor de multiplicación infinito k_∞ del núcleo. El elemento combustible está compuesto por un arreglo de 37 barras individuales que contienen las pastillas de dióxido de uranio recubiertas por un cladding de zircalloy.

En los reactores de agua pesada la parada rápida del reactor se realiza mediante la inserción de las barras de control por gravedad tal como en los reactores de agua liviana. Pero el segundo sistema de extinción consiste en la inyección rápida de una solución líquida absorbente de neutrones en el

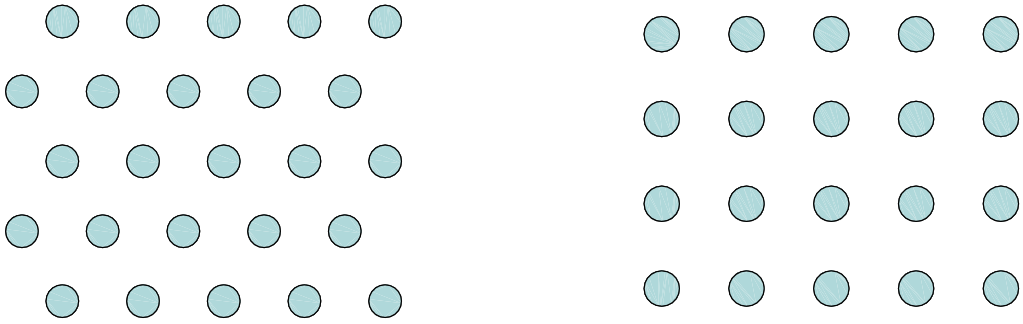


(a) Esquema de dos pasos estimando la primero reactividad $\rho_b(t)$ debida a la inyección de boro y luego incorporándola a las ecuaciones de cinética puntual.



(b) Esquema completamente acoplado utilizando cinética neutrónica espacial.

Figura 1.4.: Esquema de acople propuesto para el modelado de accidentes que involucran la actuación del sistema de inyección de boro de emergencia en la Central Nuclear Atucha I. La primera alternativa es calcular la reactividad negativa debida a la inyección de boro y luego incorporar $\rho_b(t)$ a la reactividad total de las ecuaciones de cinética puntual. La segunda consiste en un esquema completamente acoplado utilizando cinética neutrónica espacial. En cualquier caso, la forma de incorporar la distribución espacio-temporal de concentración de boro en el moderador al código neutrónico es la misma.



(a) Atucha (canales verticales) (b) CANDU (canales horizontales)

Figura 1.5.: Arreglo de canales en el plano perpendicular a la dirección axial en los reactores de agua pesada que operan en la Argentina. En ambos casos los canales están inmersos en un tanque que contiene agua pesada que actúa como moderador de los neutrones que nacen en los elementos combustibles alojados en los canales. Ambas figuras están representadas en la misma escala espacial, por lo que se puede observar la diferencia en los pasos y en los radios de los canales.

tanque del moderador, que es un componente único de este tipo de reactores. En particular, para el caso de Atucha I y II se emplea ácido deuterobórico.

1.2.6. Dos son compañía, tres son multitud

Podemos estudiar la mayoría de los casos que componen el Capítulo 15 "Análisis de accidentes" del FSAR utilizando cinética neutrónica puntual separando las contribuciones individuales $\rho_x(t)$ debido al efecto x (barras de control, temperaturas, densidades, xenón, etc.) y luego sumándolas algebráicamente para obtener una reactividad total $\rho(t)$. Sin embargo, en algunos casos tales como los accidentes con pérdida de refrigerante, es importante que consideremos efectos espaciales al presentarse una retroalimentación compleja debido tanto a las características termohidráulicas como neutrónicas de este tipo de reactores. Ambas situaciones son tenidas en cuenta en el esquema acoplado propuesto, que ilustramos en la figura 1.4. En él se involucra a un código de planta, a un modelo de la lógica de control y protección del reactor y a un código de cinética espacial capaz de incorporar distribuciones espacio-temporales de propiedades, en particular concentración de boro en el moderador, calculadas a partir de técnicas de dinámica de fluidos computacional,[5] conocidas por sus siglas como CFD.

1.2.7. La gloriosa discretitud del alfabeto

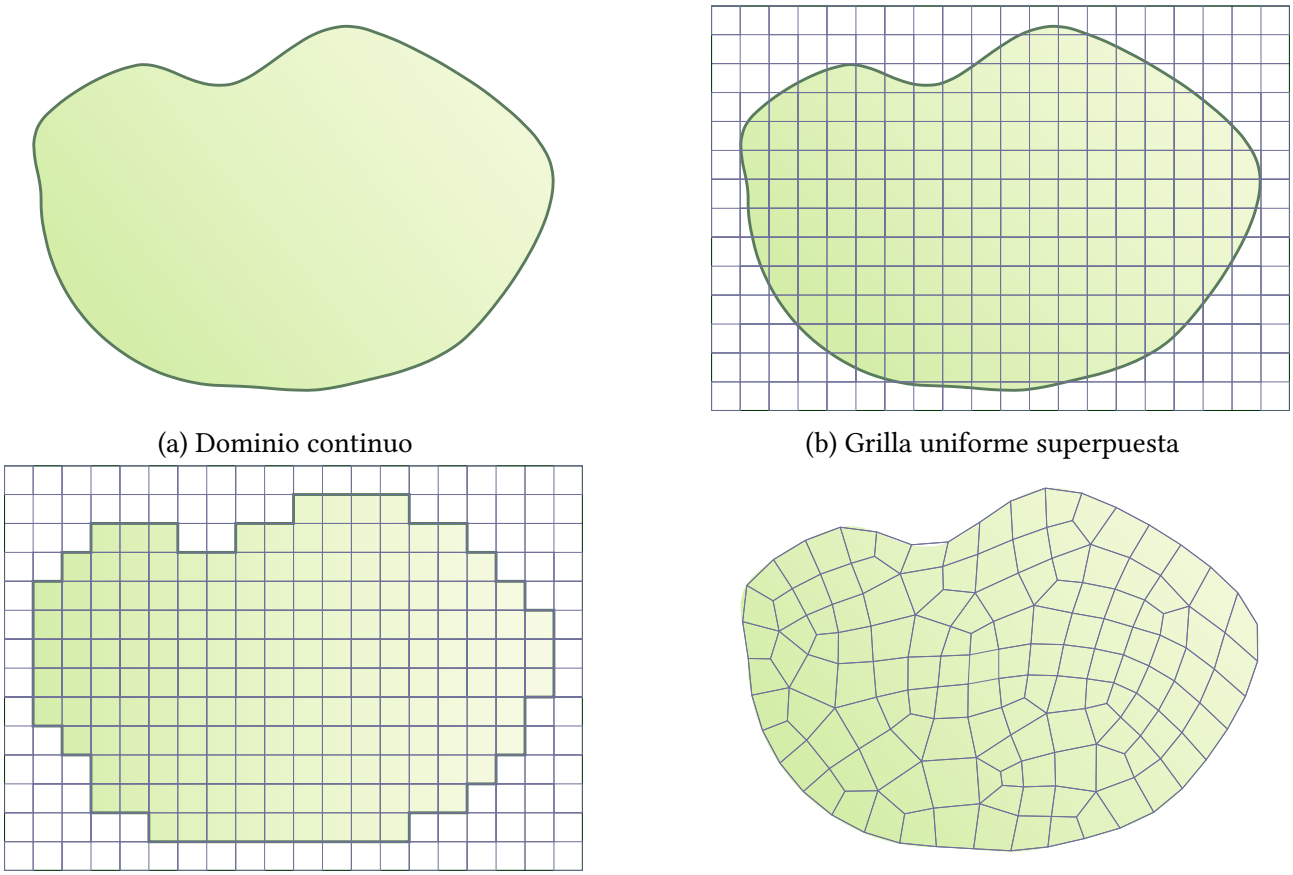
El punto principal de esta tesis es que las herramientas de neutrónica de núcleo al final del día lo que hacen es utilizar computadores digitales para resolver ecuaciones diferenciales en derivadas par-

[5]Del inglés *Computational Fluid Dynamics*.

ciales. Para ello es necesario discretizar el dominio espacial de la ecuación diferencial en derivadas parciales para obtener una cantidad finita de ecuaciones diferenciales ordinarias. La forma de discretizar el dominio depende de la formulación espacial discretizada. Los tres esquemas más comunes son

1. diferencias finitas (FDM)
2. volúmenes finitos (FVM)
3. elementos finitos (FEM)

Dado un dominio espacial continuo (figura 1.6a), esencialmente se puede proceder de dos maneras diferentes. O bien se superpone una grilla cartesiana (figura 1.6b) y luego se ajusta el dominio a la grilla para obtener una malla estructurada (figura 1.6c). O bien se aplican técnicas de mallado no estructurado para obtener una malla que permita representar la geometría original con mucha mayor precisión para la misma cantidad de entidades discretas (figura 1.6d).



(a) Dominio continuo

(b) Grilla uniforme superpuesta

(c) Malla estructurada

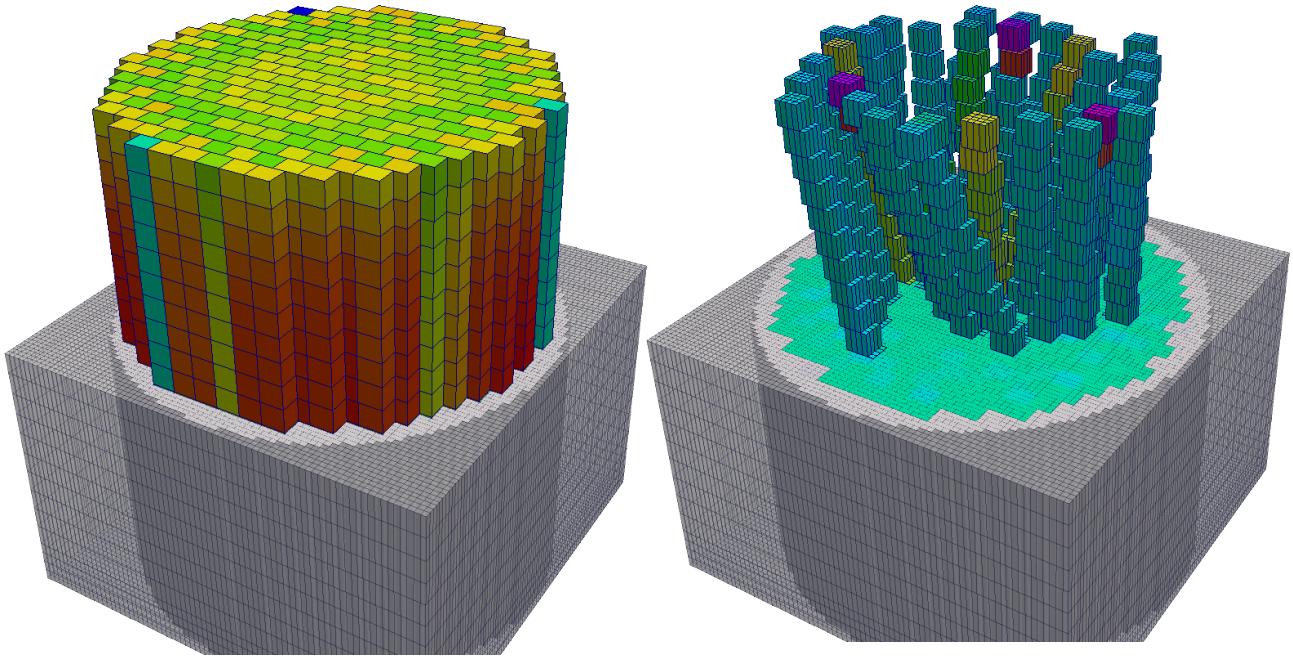
(d) Malla no estructurada

Figura 1.6.: Discretizaciones estructurada y no estructurada de un dominio espacial arbitrario.

La principal diferencia técnica entre estas dos clases de mallas reside en que en el primer caso la topología se da implícitamente con una cantidad mínima de información, como por ejemplo número de celdas en cada dirección cartesiana o un vector de tamaños de celdas en cada dirección si la malla no es uniforme. En cambio, en el caso de mallas no estructuradas es necesario dar una lista explícita y completa indicando qué nodos definen qué celdas para poder obtener la topología y saber, por

ejemplo, cómo es la conectividad de las celdas. Una forma eficiente de proveer esta conectividad es construir un grafo dirigido acíclico[6] (DAG) [7], [32].

La mayoría de las herramientas de neutrónica a nivel de núcleo utilizadas en la industria nuclear mundial soportan solamente mallas estructuradas figura 1.7. En particular, todas las herramientas de neutrónica a nivel de núcleo empleadas en el análisis de seguridad de reactores tipo Atucha utilizan mallas estructuradas para resolver la ecuación de difusión de neutrones. Por lo tanto, cuando hablemos de neutrónica en lo que resta del capítulo solamente aparecerán mallas estructuradas hasta que discutamos las propuestas de esta tesis de doctorado en la sección 1.4.



(a) Malla de representación

(b) Malla de cálculo

Figura 1.7.: Mallas *estructuradas* del código neutrónico utilizado en el esquema acoplado propuesto de la figura 1.4 para Atucha II [65].

1.2.8. Multi-física multi-escala

En cualquiera de los dos casos ilustrados en la figura 1.4 para la evaluación de la neutrónica asociada a la actuación del sistema de inyección de boro de emergencia, el código neutrónico resuelve la ecuación de difusión de neutrones en el núcleo con secciones eficaces macroscópicas homogeneizadas espacialmente a nivel de canal refrigerante individual y condensadas a dos grupos de energía. Es decir, la celda unitaria que se resuelve en el nivel de cálculo de celda en el esquema multi-escala usual (sección 2.5) contiene un canal refrigerante con las barras que componen el elemento combustible y una porción de moderador asociada a dicho canal (figura 1.8).

El código de núcleo trabaja con dos mallas (o retículas según la nomenclatura propuesta por el autor original del código), ambas *completamente estructuradas*: una de representación y una de cálculo

[6]Del inglés *directed acyclic graph*.

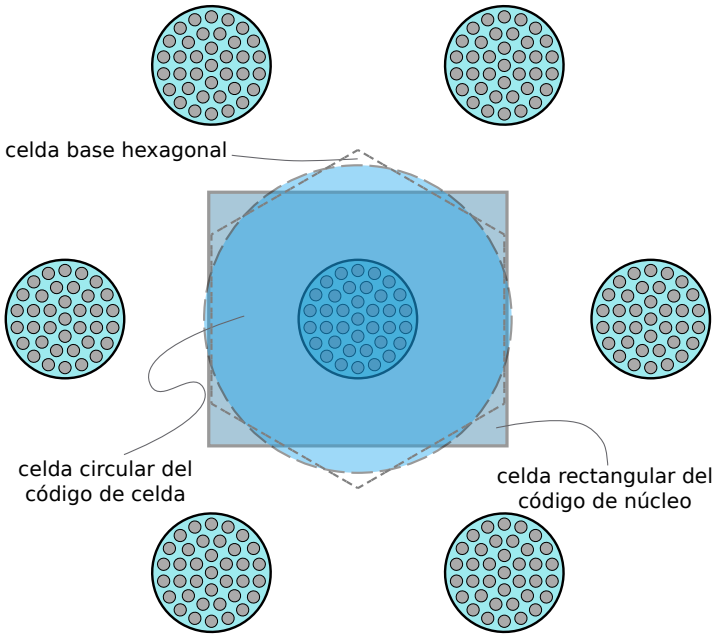


Figura 1.8.: Celda neutrónica unitaria para el núcleo de Atucha I [78]. El cálculo de celda se realiza sobre un círculo de radio equivalente a la celda hexagonal geométrica. El cálculo de núcleo utiliza las secciones eficaces homogeneizadas a dos grupos y las aplica a una celda rectangular. Esta última celda es la que define la malla de representación de la figura 1.7a.

como ilustramos en la figura 1.7. En la primera es donde se definen las secciones eficaces macroscópicas que dependen de las propiedades medias de la celda: quemado y temperatura de combustible, temperaturas y densidades de moderador y refrigerante, concentración de boro en moderador y refrigerante, concentración de xenón 135 en el combustible, etc. En la segunda, que es una subdivisión de la primera, es donde se resuelve numéricamente la ecuación de difusión de neutrones a partir de las secciones eficaces definidas sobre la malla de representación.

Para el caso particular de Atucha I, en la malla de representación se define la ubicación espacial de los 253 canales. La malla de cálculo resulta de dividir sobre el plano x-y el rectángulo asociado a cada celda en $n \times n$ rectángulos más pequeños (por la geometría del arreglo de canales n debe ser una potencia de dos) y la longitud activa h del núcleo en una cantidad m de celdas axiales. En el equipo de trabajo entonces se dice que un cálculo se realiza con una malla de $2 \times 2 \times 20$, $4 \times 4 \times 80$, etc. La figura 1.7b muestra una malla de cálculo de $4 \times 4 \times 20$.

1.2.9. Dos por uno en D_2O

Durante la interacción con los consultores extranjeros en la etapa de evaluación de la inyección de boro en Atucha II hemos identificado que una discretización espacial gruesa tiende a sobrestimar la reactividad negativa introducida de forma inaceptable debido al efecto de dilución de secciones eficaces que repasamos en el capítulo de resultados (ver sección 5.6). En efecto, consideremos una pequeña gota de ácido deuterobórico con una gran concentración de boro, digamos 2000 partes por millón, que en algún instante se encuentra dentro de una de las celdas sobre las cuales se homogeneizan las secciones eficaces macroscópicas, como ilustramos en la figura 1.9, y supongamos que la

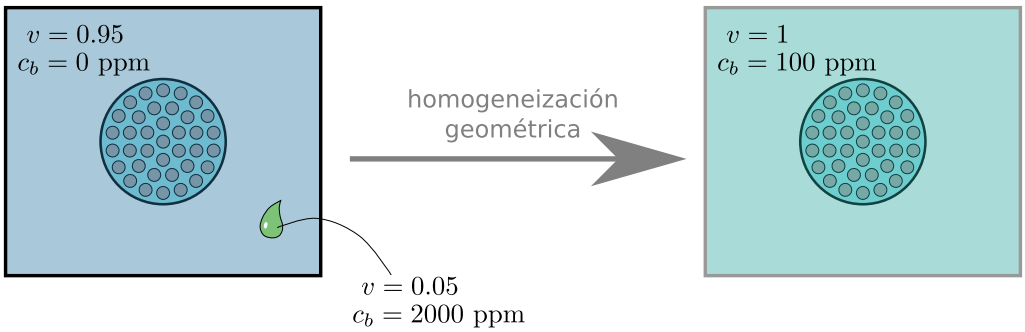


Figura 1.9.: Efecto de dilución geométrica de secciones eficaces. Una pequeña gota de absorbente al ser diluida en el volumen de una celda mucho mayor utilizando sólo relaciones geométricas resulta en secciones eficaces homogeneizadas excesivamente absorbentes. Un absorbente negro del 5% del volumen de una celda transforma la celda completa en un absorbente casi negro. Esto es, una absorción total de los neutrones que llegan a una fracción del volumen se transforma en una absorción "casi" total pero ahora de los neutrones que aparecen en la celda. Si el mundo fuese lineal esto no importaría, pero la dependencia de la absorción de neutrones es muy diferente a 0 ppm que a 100 o 2.000 ppm.

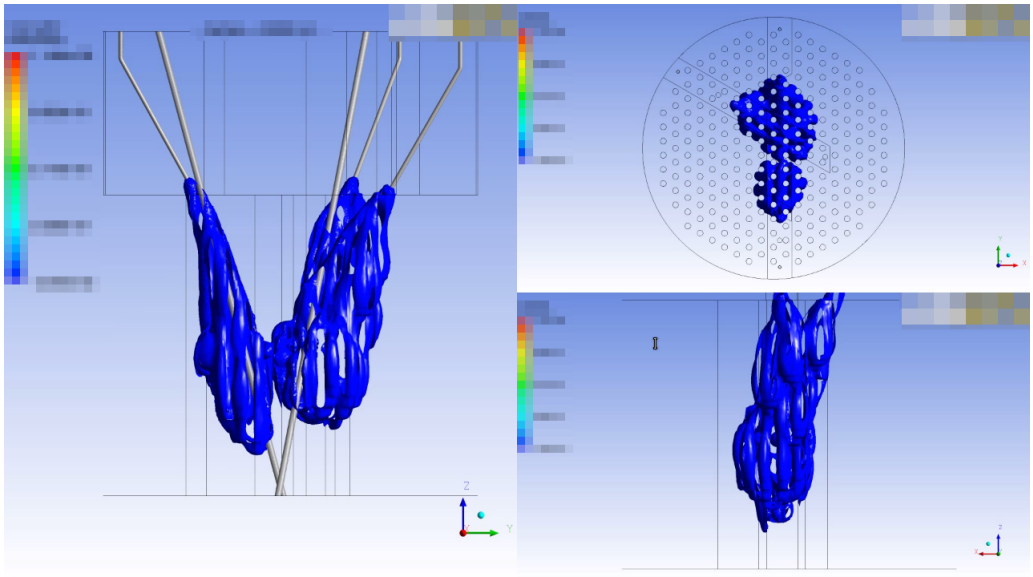


Figura 1.10.: Cálculo fluidodinámico de la evolución temporal de la pluma de boro en el tanque del moderador de Atucha I realizada por ingenieros de NA-SA con técnicas CFD sobre una malla no estructurada de aproximadamente 4.5 millones de celdas [74]. Se pueden observar los huecos en la distribución espacial generados por la presencia de los canales.

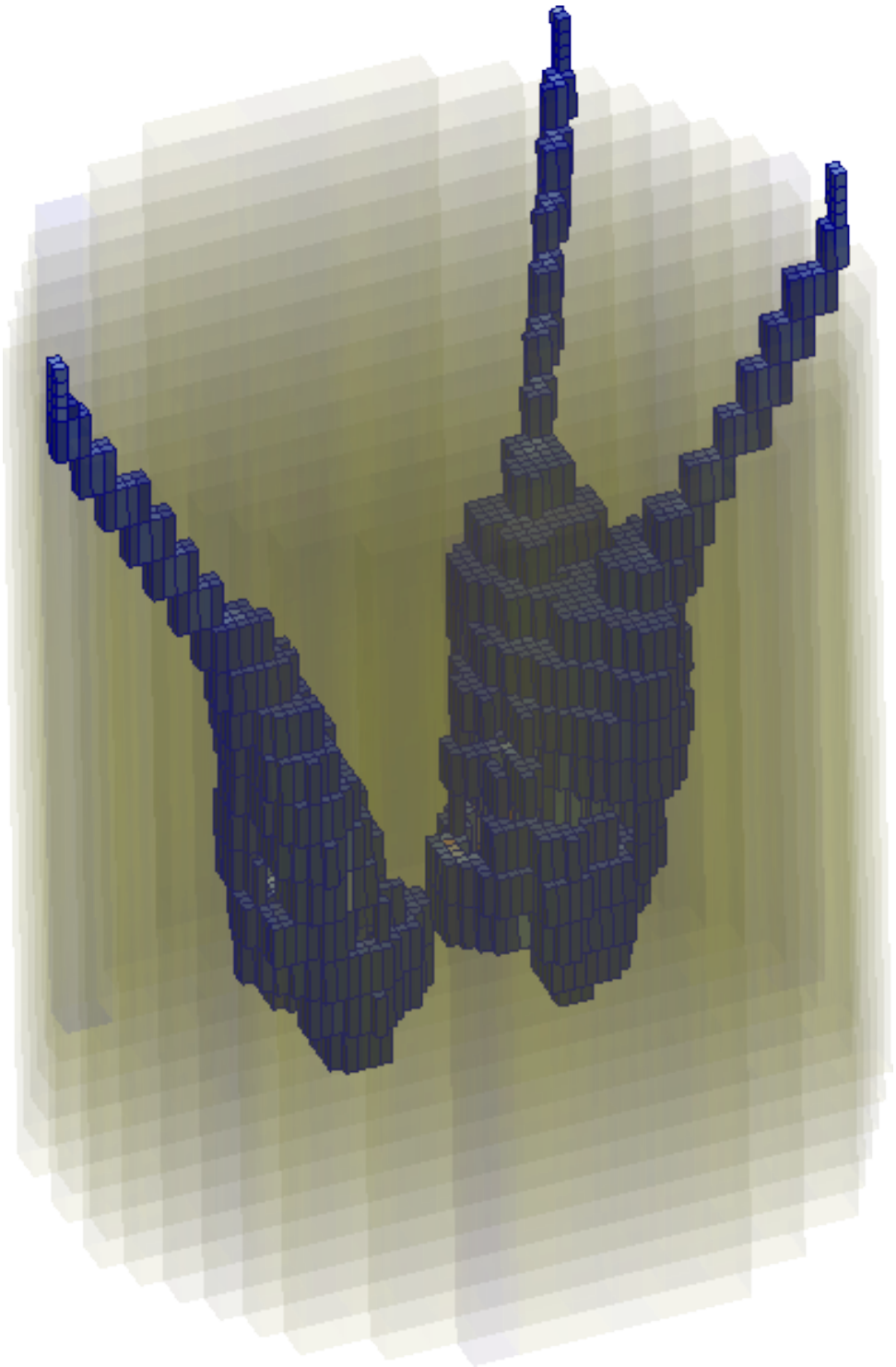
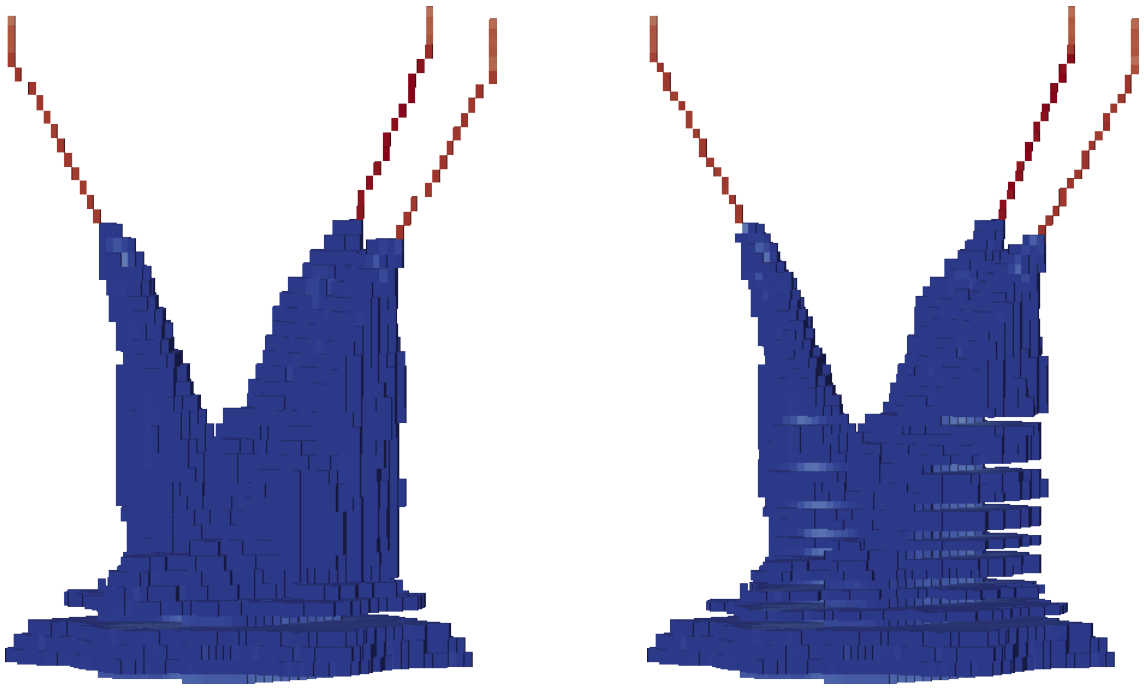


Figura 1.11.: Mapeo de la distribución instantánea de boro a una malla de cálculo del código neutrónico de núcleo de $4 \times 4 \times 20$ ($\sim$ 200.000 celdas) [74]. No es posible observar la presencia de los canales.

gota ocupa el 5% del volumen de la celda. Para poder avanzar un paso del cálculo cinético-espacial de núcleo debemos asignarle secciones eficaces mascroscópicas a la celda que contiene la pequeña gota de boro, a partir de cálculos paramétricos de nivel de celda en los cuales conocemos cómo varían las secciones eficaces en función de los parámetros termohidráulicos (temperaturas y densidades) y de la concentración de venenos (xenón y boro) de la celda. Como el boro no está uniformemente distribuido, debemos obtener un valor medio que proponemos calcular como un promedio de las concentraciones de boro de la gota y del resto de la celda pesado con los volúmenes relativos. Esta propuesta no conserva ritmos de reacción, pero no hay mejores alternativas dado que existe un único parámetro de boro para toda la celda. Para el caso de la figura 1.9, la concentración media de boro de la celda sería 100 ppm, resultando en una absorción casi negra para toda la celda en lugar de una absorción completamente negra sólo en el 5% del volumen. La alternativa a la dilución geométrica sería homogeneizar de forma tal no de mantener la relación de volúmenes sino los ritmos de reacción. Esto implicaría tener que realizar un nuevo cálculo de celda para cada una de las celdas de la malla de representación para cada instante de tiempo teniendo en cuenta la geometría real de la gota, lo que de hecho está fuera de las posibilidades de las cadenas de cálculo actuales, pero podría llegar a ser un esquema alternativo. Sin embargo, como discutimos más adelante, aún existen otros inconvenientes en la formulación que no pueden ser salvados de esta forma.



(a) $4 \times 4 \times 40$ (b) $4 \times 4 \times 60$

Figura 1.12.: Distribución de boro para un cierto instante mapeada desde la malla de CFD a la malla de cálculo neutrónico de 4×4 con (a) 40 celdas axiales y (b) 60 celdas axiales [81]. En el segundo caso con las celdas más pequeñas, el mapeo de malla no estructurada (CFD) a malla estructurada (neutrónica) no puede reproducir los resultados en las zonas donde la malla de CFD no está suficientemente refinada. El resultado es que en forma espuria se remueve boro del tanque del moderador para mallas de cálculo neutrónico demasiado finas.

Para reducir este efecto y además poder realizar un estudio de convergencia de malla hemos decidido

extender el código de núcleo para permitir que la definición de la distribución instantánea de boro pueda realizarse también sobre la malla de cálculo, en lugar de hacerlo sólo sobre la de representación como el resto de los parámetros (temperaturas, densidades, etc.) [65]. En este caso, a partir de resultados fluidodinámicos es que incorporamos la pluma de boro al esquema de cálculo acoplado mapeando explícitamente (con un código desarrollado *ad-hoc*) la malla del código CFD (figura 1.10, $\sim$ 4.5 millones de celdas) a la malla de cálculo del código neutrónico (figura 1.11, $\sim$ 200.000 celdas para $4 \times 4 \times 20$).

Observación. En la sección 5.1 del capítulo 5 mostramos el enfoque propuesto para tratar mallas no conformes a partir de las lecciones aprendidas. En la sección B.3.2.2 resolvemos un problema termo-mecánico usando el mismo esquema.

1.2.10. Celdas refinadas

Dado que el efecto de dilución de secciones eficaces macroscópicas se reduce drásticamente con el tamaño de la celda, hemos realizado estudios de convergencia de malla [81] con el doble objetivo de estimar los errores cometidos y para extrapolar las reactividades obtenidas a una discretización espacial infinitesimal. Sin embargo, hemos encontrado que no siempre es conveniente trabajar con mallas de cálculo neutrónico demasiado finas. En la figura 1.12 mostramos el resultado de mapear una cierta distribución de concentración de boro para dos mallas de cálculo similares, una con 40 celdas axiales y otra con 60. Debido a que el cálculo CFD se realiza sobre una malla no estructurada, ésta se ha diseñado de forma tal de ser más refinada en ciertas ubicaciones de interés dentro del tanque del moderador. En algunas zonas donde esta malla es más gruesa, sucede que las celdas de CFD son más grandes que las celdas de cálculo neutrónico. El resultado es que, para mallas neutrónicas demasiado finas, pueden quedar algunas celdas sin tener asignada una concentración de boro ya que todo el boro de la celda CFD podría asignarse a una celda neutrónica vecina. De esta manera, a partir de un cierto tamaño de malla neutrónica, si bien se disminuye el efecto de la dilución de secciones eficaces sucede que en forma numérica se remueve boro del tanque del moderador. Estos dos efectos tienen consecuencias opuestas, y su magnitud no es fácil de evaluar. Una alternativa para paliar este problema sería plantear un método de conversión entre la celda de CFD y la de cálculo neutrónico que tenga en cuenta estos casos particulares. Sin embargo, como discutimos más, aún existen otros inconvenientes en la formulación que no pueden ser salvados de esta forma.

En la figura 1.10, podemos observar que la nube de boro calculada en la malla de CFD avanza sólo en el tanque del moderador. Es decir, los canales individuales forman parte de la frontera del dominio fluidodinámico. En la malla neutrónica, los canales están embebidos en un arreglo de paralelepípedos que no son capaces de reproducir su geometría cilíndrica. En efecto, en la figura 1.13 ilustramos este concepto suponiendo que el frente de la pluma de boro hace contacto con un canal refrigerante. Al homogeneizar geométricamente, la celda 1 tendrá asignada una concentración de boro $c_{b1} = 1000$ ppm ya que se encuentra en su totalidad dentro de la nube. Las celdas 2 y 3 tendrán alguna cierta concentración de boro entre 0 y 1000 ppm. Estas celdas sufrirán en alguna medida el efecto de dilución de secciones eficaces ya discutido. Pero la situación en la celda 4 es más compleja aún, porque la concentración c_{b4} asignada supone que todo el volumen de la celda está compuesto por moderador, cuando la mayor parte consiste en una mezcla de combustible y refrigerante. Más aún, el hecho de que la concentración c_{b1} sea diferente de la nominal (i.e. $c_b = 0$) hará que el código de núcleo modifique todas las secciones eficaces macroscópicas de la celda 1, incluso las relacionadas

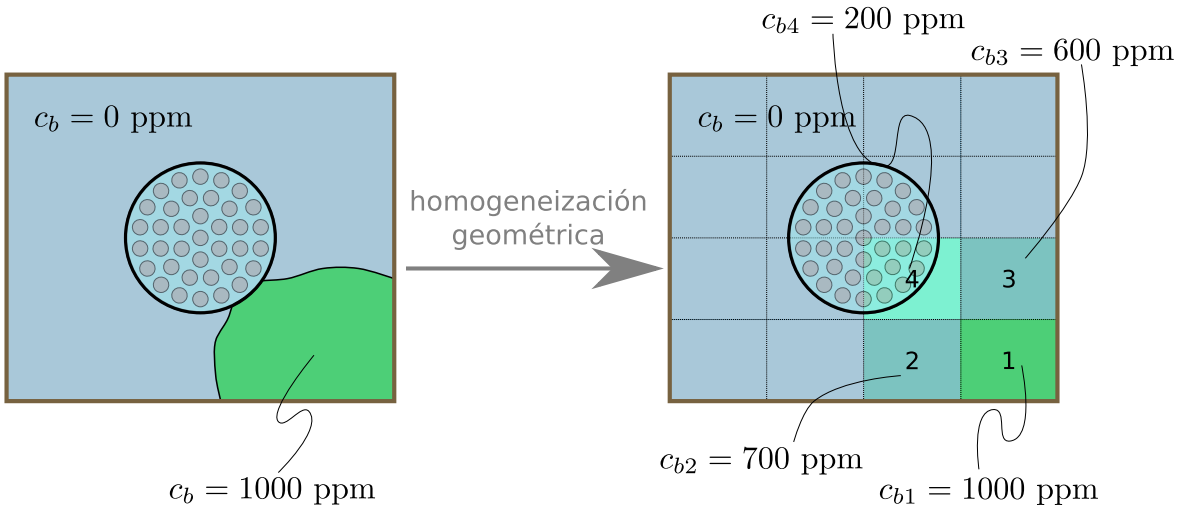


Figura 1.13.: Cuando el frente de la pluma de boro llega a un canal refrigerante, la homogeneización geométrica asigna concentraciones de boro según las relaciones de los volúmenes de las celdas con respecto a la pluma discretizada en la malla de CFD. Como el moderador no está separado de la mezcla combustible más refrigerante, se producen inconsistencias en las secciones eficaces asignadas a cada celda de cálculo.

a la fisión cuando en realidad no hay materiales fisibles ni fisionables en ella por el simple hecho de tener secciones eficaces macroscópicas homogeneizadas teniendo en cuenta una celda con "un poco de combustible y un poco de moderador".

1.2.11. Neutrones difundidos

Tanto el código neutrónico de núcleo empleado en el esquema acoplado propuesto para actualizar los transitorios del Capítulo 15 del FSAR de Atucha I ilustrado en la figura 1.4 como el código utilizado por los expertos internacionales para estudiar la inyección de boro durante el licenciamiento de Atucha II resuelven la ecuación de difusión de neutrones a dos grupos de energía. En esta formulación, se supone que la corriente neta de neutrones es proporcional al gradiente del flujo escalar a través de un cierto coeficiente de difusión. Esta ecuación es una aproximación que podemos deducir a partir de la ecuación de transporte de neutrones (de hecho la deducimos matemáticamente en esta tesis en la sección 2.4). La validez de la aproximación es tanto mejor mientras más ciertas sean las famosas seis suposiciones de la página 125 del libro clásico de Lamarsh [31]:

1. el medio es infinito
2. el medio es uniforme, de forma tal que todas las secciones eficaces son constantes, independientemente de la posición
3. no hay fuentes de neutrones
4. el scattering[7] es isotrópico en el sistema de coordenadas del *laboratorio*[8]
5. el flujo neutrónico es una función levemente dependiente de la posición
6. el flujo neutrónico no es un función del tiempo.

[7] El término castellano "dispersión" nos parece menos apropiado que el original en inglés scattering.
[8] De *la planta* o *la central* serían mejores términos para el caso tratado en este trabajo.

Si bien estas hipótesis pueden ser relajadas y aún así poder suponer que la corriente neta de neutrones es proporcional al gradiente del flujo escalar, esto deja de ser cierto en presencia de materiales muy absorbentes. Este hecho es especialmente importante cuando hay interfaces entre materiales en donde se dan grandes discontinuidades en las secciones eficaces, que es justamente el objetivo de la evaluación del segundo sistema de extinción: el avance de una pluma de un absorbente neutrónico (ácido deuterobórico) a través de un medio difusivo (el agua pesada contenida en el tanque del moderador).

En el capítulo 2 derivamos primeramente la ecuación de transporte de neutrones a partir de la conservación de neutrones. Luego derivamos la ecuación de difusión a partir de la de transporte y mostramos detalladamente las razones matemáticas de las aproximaciones necesarias para llegar a la ley de Fick para neutrones. Pero este enfoque solamente involucra aproximaciones del orden de "el cociente $a/b \ll 1$" o "despreciando términos de orden superior" sin tener parámetros numéricos para su evaluación. La implicación física de estas suposiciones y aproximaciones solamente puede ser evaluada resolviendo un mismo problema con las dos ecuaciones y evaluando la diferencia obtenida en las soluciones. Este es justamente uno de los aportes de esta tesis de doctorado.

La condensación y homogeneización de secciones eficaces que realiza el código de celda tiende a reducir las variaciones espaciales propias de configuraciones heterogéneas de los núcleos de los reactores nucleares. Pero tal como hemos discutido, para reducir los efectos de la dilución del boro es necesario reducir los tamaños de las celdas de cálculo. Esta solución, además de no ser completamente efectiva para contrarrestar la demasiada inserción de reactividad negativa, configura una situación en la cual la ecuación de difusión deja de ser válida. En particular, las condiciones 2 y 5 dejan de cumplirse y no sólo no podemos evaluar la magnitud sino que ni siquiera podemos conocer el signo del error cometido.

1.3. Limitaciones de la formulación actual

El enfoque propuesto para evaluar la efectividad del segundo sistema de apagado de un reactor tipo Atucha involucra los siguientes pasos:

1. Realizar un cálculo tipo CFD para obtener la distribución espacio-temporal de concentración de ácido deuterobórico sobre una malla detallada teniendo en cuenta la geometría de
    - el tanque del moderador
    - los canales combustibles
    - los tubos guía de barras de control
    - las lanzas de inyección de boro
2. Convertir dicha distribución a una malla estructurada para que esa información pueda ser tenida en cuenta por las herramientas neutrónicas disponibles.
3. Calcular la dependencia de las secciones eficaces macroscópicas del moderador en función de la concentración de boro compatibles con la aproximación de difusión.
4. Realizar un cálculo de difusión de neutrones a nivel de núcleo tomando la información obtenida en los puntos 2 y 3 como entrada.

Tanto la metodología como los estudios propiamente dichos han sido aprobados por la Autoridad Regulatoria Nuclear para su aplicación a Atucha I y, de hecho, es en varios aspectos técnicos superior al implementado por expertos internacionales para Atucha II. Sin embargo, esta metodología presenta al menos tres inconvenientes:

i. Una malla de cálculo neutrónico muy gruesa con respecto a la malla de CFD produce una dilución de secciones eficaces absorbentes que hace que se sobrestime la reactividad negativa introducida por el boro, aunque un refinamiento excesivo de la malla de cálculo neutrónico remueve boro numéricamente en forma espuria introduciendo un efecto opuesto.[9]

ii. Los canales no se pueden representar en forma satisfactoria con un malla de cálculo neutrónico estructurada basada en paralelepípedos, y para tamaños de celda menores a la mitad de la distancia entre canales el código termina modificando secciones eficaces de fisión homogeneizadas en función de la concentración de boro en ubicaciones donde no hay combustible sino sólo moderador.

iii. No se cumplen enteramente las condiciones de validez de la ecuación de difusión de neutrones, por lo que el cálculo neutrónico de núcleo tiene una incerteza demasiado grande.

Aún cuando es posible que los resultados de ingeniería obtenidos sean suficientemente exactos y precisos, no es posible determinar cuantitativamente dicha exactitud y precisión si no se dispone de un método objetivamente más exacto y preciso contra el cual poder comparar los resultados.

1.4. Las propuestas de esta tesis

La motivación fundamental de esta tesis parte del tercer punto de la sección anterior, y luego continúa subiendo hacia los otros dos puntos, encadenando varias ideas en un mismo hilo conductor: desarrollar una herramienta objetivamente más precisa para poder verificar la precisión y exactitud de las aproximaciones involucradas en cálculos neutrónicos a nivel de núcleo.

Está claro que resolver una formulación de la ecuación de transporte basada en el método de ordenadas discretas S_N arroja resultados más precisos que la ecuación de difusión. Ahora bien, el principal problema de S_N es que el tamaño del problema discretizado escala como el producto de

a. la cantidad de grupos de energía (sección 3.2),
b. la cantidad $N \cdot (N + 2)$ de direcciones de vuelo de los neutrones en la formulación S_N (sección 3.3),
c. la cantidad de incógnitas espaciales, es decir el número de nodos o celdas de la malla espacial (sección 3.4),

A su vez, los recursos computacionales necesarios para resolver el problema, esencialmente tiempo de procesador y memoria, escalan con una velocidad más que lineal con el producto de los tres puntos mencionados, usualmente entre $O(n \log n)$ y $O(n^2)$. Esto hace que para casos con mallas de interés de ingeniería sea imposible emplear una única computadora digital para resolver el problema. En efecto, aún cuando un tiempo de procesamiento arbitrariamente grande pueda ser aceptado

[9] Estrictamente hablando este efecto puede ser corregido, pero el problema de la escalabilidad persiste.

desde un punto de vista de gerenciamiento de proyectos, la escala de la memoria requerida no permitiría la flexibilidad suficiente para realizar los estudios de convergencia de malla, de energía y de direcciones necesarios para estudiar la verificación tanto de la herramienta de transporte por S_N en sí como de la comparación con formulaciones más simplificadas basadas en difusión. Es necesario entonces diseñar un esquema numérico de resolución de ecuaciones diferenciales que pueda escalar con el tamaño del problema a resolver. Es decir, la memoria le impone un límite técnico al proyecto. Para ello, la herramienta computacional utilizada en el proyecto debe ser paralelizable, flexible y extensible:[10]

$$\text{difusión} \rightarrow \text{muy simplificado} \Rightarrow S_N \rightarrow \text{escala muy rápido} \Rightarrow \text{esquema} \begin{cases} \text{paralelizable} \\ \text{flexible} \\ \text{extensible} \end{cases}$$

Paralelizable de forma que el problema se pueda resolver con un cierto número de computadoras digitales trabajando en conjunto de forma tal que los recursos computacionales (esencialmente la memoria) sean suficientes para que cada una de estas computadoras pueda tratar una parte del problema.

Una condición necesaria para poder implementar este tipo de paralelización es poder dividir el dominio espacial de forma tal de minimizar la cantidad de información que deben compartir cada una de las computadoras con el resto. Esto implica resolver un problema de descomposición de dominio (DDM) que involucra teoría de grafos (DAG), lo que a su vez necesita la topología explícita de la malla.

Este requerimiento concuerda con la necesidad de utilizar mallas no estructuradas para poder modelar geometrías complejas en reactores PHWR, a saber

- moderador separado de refrigerante en canales circulares
- barras de control inclinadas
- lanzas de inyección de boro inclinadas
- plumas de ácido deuterobórico transitorias no triviales

Las mallas no estructuradas incluso permiten también

- modelar reactores de investigación sin sufrir el "efecto escalera" de la figura 1.6c
- evitar efectos numéricos tales como la dilución de secciones eficaces en los extremos de las barras de control
- facilitar la resolución de problemas tipo benchmark en geometrías cilíndricas o esféricas
- resolver problemas abstractos tales como dominios con forma de conejo o una transición entre un cubo y una esfera
- utilizar la misma malla para CFD y para neutrónica

Flexible en el número de computadoras a utilizar dependiendo del tamaño del problema y en la forma de definir tanto la entrada como la salida de datos, haciendo especial énfasis en la necesidad de trabajar en entornos de nube pública.

[10] El símbolo $\rightarrow$ quiere decir "implica" y el símbolo $\Rightarrow$ quiere decir "luego se necesita".

Un problema de tamaño arbitrario en principio requeriría una cantidad también arbitraria de computadoras para ser resuelto. Una herramienta computacional diseñada para correr en un cluster de cálculo de tamaño fijo no cumpliría esta condición, por lo que debe ser posible explotar la oferta de servidores públicos en la nube. Luego la herramienta computacional debe ser diseñada desde el comienzo para operar en la nube (i.e. cloud-first) en lugar de que sea solamente "posible" su ejecución en la nube (i.e. cloud-friendly). Hay sutiles pero importantes diferencias entre estos dos conceptos, discutidos en la sección 4.4.6, tales como

- necesidad de proveer una interfaz pública tipo RESTful API
- capacidad de definir, lanzar y post-procesar cálculos con interfaces web
- posibilidad de reportar el estado del cálculo y, eventualmente, errores en forma remota

Extensible para poder modificar o agregar modelos matemáticos que eventualmente ayuden a mejorar la calidad, precisión y exactitud de los resultados obtenidos.

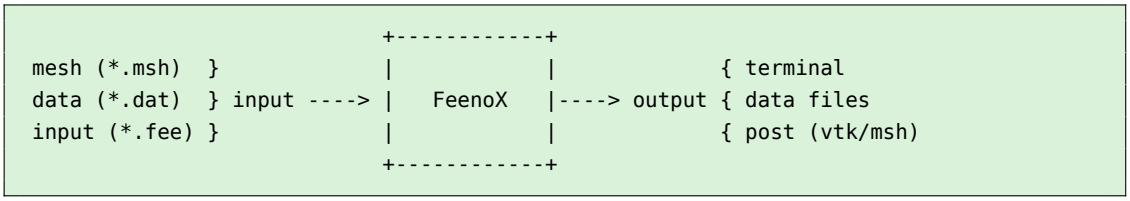
Está claro que teniendo acceso al código fuente de una herramienta computacional, en principio siempre es posible modificar y/o agregar nuevas formulaciones y/o modelos matemáticos. Sin embargo, el concepto de extensibilidad implica que se hayan tenido en cuenta posibles mecanismos de extensión en el diseño de la arquitectura del código de forma tal que el esfuerzo necesario para lograr extender la funcionalidad sea razonable. Más aún, bajo el espíritu académico de un trabajo de doctorado, la herramienta debe calificar como *software libre* en el concepto de la Free Software Foundation de forma tal de que cualquier investigador o profesional pueda modificarla y/o extenderla para poder resolver problemas planteados como ecuaciones diferenciales parciales de la mejor manera posible.

La propuesta de esta tesis es entonces desarrollar una herramienta computacional que esencialmente satisfaga estas condiciones. Es por eso que:

1. Los esquemas numéricos desarrollados a lo largo del Capítulo 3 para resolver las ecuaciones de transporte y difusión de neutrones introducidas en el capítulo 2 se basan en formulaciones basadas en elementos finitos, que son intrínsecamente compatibles con mallas no estructuradas.

2. Tal como discutimos en la sección 1.1, el primer requerimiento de la herramienta computacional desarrollada es que sea cloud first. Los apéndices A y B describen los requerimientos y las especificaciones desde el punto de vista de desarrollo de software. En resumen, la herramienta…

   - es libre y abierta distribuida bajo licencia GPLv3+ (sección B.1)
   - sigue la filosofía de programación Unix [47]. Estrictamente hablando es un filtro de Unix que funciona como una función de transferencia (sección B.1.2) entre



a. uno o más archivos de entrada de texto plano que definen completamente la entrada, y
b. cero o más archivos de salida (posiblemente incluyendo `stdout`) con los resultados solicitados:

- no escribe (y muy probablemente ni siquiera calcule) un resultado si éste no se pide explícitamente como una salida (sección B.3.2)
- los archivos de entrada (sección B.3.1) deben…
  - reproducir la estructura de la definición del problema a resolver que haría un profesional humano.
  - ser simples para problemas simples.
  - ser parecidos para problemas parecidos.
  - separar la definición del problema continuo a resolver de la discretización espacial particular utilizada para resolverlo.
  - permitir una dependencia espacial no trivial de las propiedades de los materiales, por ejemplo
    * quemado
    * de concentración de venenos neutrónicos
    * posición de barras de control

    a través de
    * expresiones algebraicas, y/o
    * funciones definidas por puntos
      · a partir de una distribución unidimensional
      · definidos sobre una malla con topología explícita
      · dispersos en varias dimensiones sin topología
  - ser amigables con…
    * los sistemas de control de versiones distribuidos como Git
    * lenguajes de expansión de macros como M4
    * interfaces gráficas de usuario, especialmente basadas en web

3. El capítulo 4 describe en detalle la arquitectura elegida para permitir resolver ecuaciones diferenciales en derivadas parciales arbitrarias con una arquitectura donde existe un *framework* (ver definición 4.1) general y un esquema de apuntadores a función con *entry points* particulares para las diferentes ecuaciones a resolver. De hecho las ecuaciones de difusión de neutrones y transporte por el método S_N son casos particulares de otras formulaciones que la herramienta también puede resolver:

   - ecuación de Laplace/Poisson (tanto estado estacionario como transitorio)
   - conducción de calor (tanto estado estacionario como transitorio, incluyendo conductividad no lineal dependiente de la temperatura)
   - elasticidad lineal (sólo estado estacionario)

- análisis modal

La resolución de todas estas ecuaciones en derivadas parciales siguen la misma metodología con respecto al manejo de

- mallas no estructuradas
- expresiones algebraicas
- propiedades de materiales
- condiciones de contorno
- resolución de sistemas de ecuaciones
  - lineales
  - no lineales
  - transitorias
  - de autovalores
- escritura de resultados
  - generación de archivos de post-procesamiento (VTK, Gmsh)
  - evaluación de distribuciones en puntos arbitrarios del dominio
  - cálculo de escalares a partir de distribuciones
    - integrales sobre el espacio
    - cálculo de extremos y valores medios

Todos estos puntos están manejados por un framework de matemática general de la herramienta (definición 4.1). Cada ecuación diferencial particular a resolver debe ser "provista" como un subdirectorio dentro de `src/pdes` conteniendo ciertas funciones en C capaces de generar las matrices y vectores elementales de la formulación de la ecuación diferencial según el método de elementos finitos.

Finalmente, el capítulo 5 muestra algunos resultados que no podrían ser obtenidos por herramientas que no tengan al menos una de estas cuatro características distintivas del código desarrollado:

a. Filosofía Unix, integración en scripts y simulación programática
b. Mallas no estructuradas
c. Ordenadas discretas (además de difusión)
d. Paralelización en varios nodos de cálculo con MPI

Siguiendo una metodología que apunta a explicitar las especificaciones para reducir las incertezas en cuanto al alcance de proyectos de desarrollo de software industriales, el apéndice A contiene un Software Requirements Specification, que es un documento estándar en la industria del software, ficticio pero razonable que actúa como un pliego de especificaciones técnicas para una herramienta computacional genérica que bien podría haber sido escrito por una entidad pública o privada que necesite realizar cálculos de ingeniería en la nube. El apéndice B contiene el Software Design Specification de la herramienta desarrollada en esta tesis, que es el documento que de alguna manera "resuelve" las especificaciones del SRS con una propuesta en particular. Este apéndice actúa como una propuesta básica al pliego planteado por el SDS.

Terminada la explicación del *por qué* (why) pasemos entonces al *cómo* (how).

2. Transporte y difusión de neutrones

> No debemos tan sólo escribir ni tan sólo leer. Hay que acudir a la vez a lo uno y a lo otro, y combinar ambos ejercicios a fin de que, cuantos pensamientos ha recogido la lectura, los reduzca a la unidad. [...] Debemos actuar como las abejas. Las abejas revolotean de aquí para allá y van comiendo en las flores idóneas para elaborar la miel. El botín conseguido lo ordenan y distribuyen por los panales. Te recuerdo que también nosotros tenemos que imitar a las abejas y distinguir cuántas ideas acumulamos de diversas lecturas, pues se conservan mejor diferenciadas. Luego, aplicando la atención y los recursos de nuestro ingenio, debemos fundir en sabor único aquellos diversos jugos de suerte que aún cuando se muestre el modelo del que han sido tomados, no obstante, aparezca distinto de la fuente de inspiración. [...] Lo que comprobamos que realiza en nuestro cuerpo la naturaleza sin ninguna colaboración nuestra, es eso lo que tenemos que hacer con la lectura. Los alimentos que tomamos, mientras mantienen su propia cualidad y compactos flotan en el estómago, son una carga. Mas cuando se ha producido su trasformación, entonces y sólo entonces, se convierten en fuerza y sangre. Procuremos otro tanto con los alimentos que nutren nuestro espíritu. No permitamos que queden intactos cuántos hayamos ingerido para que no resulten ajenos a nosotros. Asimilémoslos. De otra suerte, irán al acervo de la memoria y no al de la inteligencia. [...] Prestémosle fiel asentimiento y apropiémonos de [lo que leemos] para que resulte una cierta unidad de muchos elementos. [...] Aunque se aprecie en ti la semejanza con algún maestro que ha calado profundamente en tu alma por la admiración, quiero que te asemejes a él como un hijo, no como un retrato. [...] ¿Cómo lograr esto te preguntas? Con una constante aplicación.
> *Lucio Séneca, Carta a Lucilio sobre la importancia de escribir, siglo I d.C.*

En este capítulo introducimos las ecuaciones que modelan el transporte de neutrones en el núcleo de un reactor nuclear con los siguientes objetivos:

1. Fijar la matemática de las ecuaciones continuas sobre las que se basa la implementación computacional detallada en el capítulo 4 de las ecuaciones de neutrónica discretizadas derivadas en el capítulo 3.

2. Declarar las suposiciones, aproximaciones y limitaciones de los modelos matemáticos utilizados.
3. Definir una nomenclatura consistente para el resto de la tesis, incluyendo los nombres de las variables en el código fuente.

No buscamos explicar los fundamentos físicos de los modelos matemáticos ni realizar una introducción para el lector lego. Para estos casos referimos a las referencias [62], [75] escritas por el autor de esta tesis y a la literatura clásica de física de reactores [15], [21], [23], [31], [34], [58]. Si bien gran parte del material aquí expuesto ha sido tomado de estas referencias, hay algunos desarrollos matemáticos propios que ayudan a homogeneizar los diferentes enfoques y nomenclaturas existentes en los libros de texto para poder sentar las bases de los esquemas numéricos implementados en el código de manera consistente. Para eso desarrollamos lógica y matemáticamente algunas ideas partiendo de definiciones básicas para arribar a expresiones integro-diferenciales que describen el problema de ingeniería que queremos resolver.

Está claro los desarrollos y ecuaciones expuestos en este capítulo son conocidos desde los albores de la física de reactores allá por mediados del siglo XX. Sin embargo, he decidido volver a deducir una vez más las ecuaciones de transporte y difusión a partir de conceptos de conservación de neutrones manteniendo muchos pasos matemáticos intermedios por dos razones:

a. a modo de escribir una especie de diario estoico como el de Marco Aurelio en el cual digiero (en el sentido de Séneca) la teoría de transporte de neutrones desarrollada a mediados del siglo XX, y
b. abrigando la esperanza de que una condensación homogeneizada[1] de varios libros de neutrónica atraiga estudiantes de grado que estén buscando una fuente de información y que, por contigüidad, éstos aprendan sobre la importancia del software libre y abierto en ingeniería que explico en la sección 4.4.1.

Para modelar matemáticamente el comportamiento de reactores nucleares de fisión debemos primero poder caracterizar campos de neutrones arbitrarios a través de distribuciones matemáticas sobre un dominio espacial $U \in \mathbb{R}^3$ de tres dimensiones.[2] Para ello, vamos a suponer que [34]

1. los neutrones son partículas clásicas, es decir, podemos conocer tanto su posición como su momento con precisión arbitraria independientemente del principio de Heisenberg,
2. los neutrones viajan en línea recta entre colisiones,
3. no existen interacciones neutrón-neutrón,
4. la colisión entre un neutrón incidente y un núcleo blanco es instantánea,
5. las propiedades de los materiales son
    a. continuas en la posición, es decir, la densidad en un diferencial de volumen es uniforme aún cuando sepamos que los materiales están compuestos por átomos individuales, e
    b. isotrópicas, es decir, no hay ninguna dirección preferencial,
6. las propiedades de los núcleos y la composición isotópica de los materiales no dependen del tiempo, y
7. es suficiente que consideremos sólo el valor medio de la distribución de densidad espacial de neutrones y no sus fluctuaciones estadísticas.

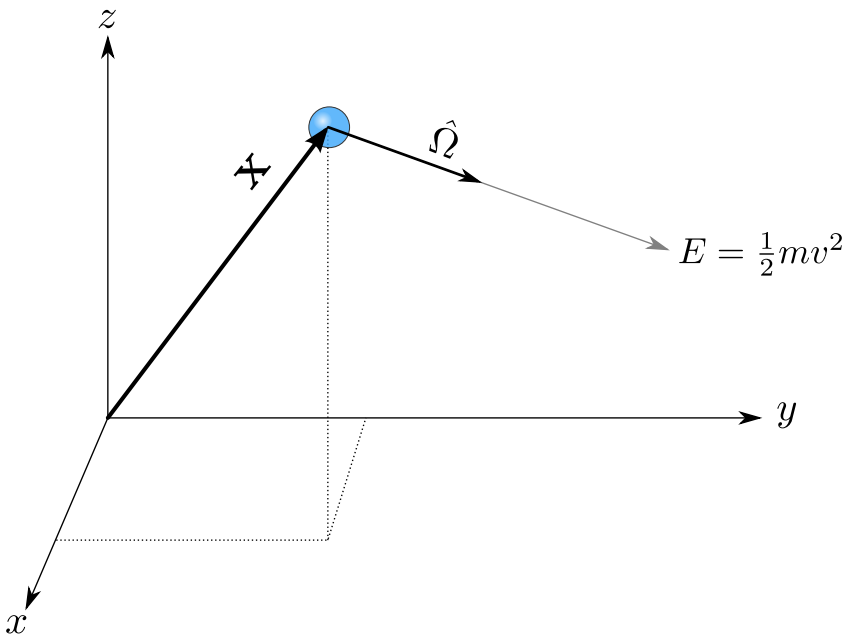


Figura 2.1.: Un neutrón individual (bola celeste, como todo el mundo sabe), en un cierto tiempo $t \in \mathbb{R}$ está caracterizado por la posición $\mathbf{x} \in \mathbb{R}^3$ que ocupa en el espacio, por la dirección $\hat{\mathbf{\Omega}} = [\Omega_x, \Omega_y, \Omega_z]$ en la que viaja y por su energía cinética $E \in \mathbb{R}$. Asumimos que podemos conocer al mismo tiempo la posición $\mathbf{x}$ y su velocidad $v \cdot \hat{\mathbf{\Omega}}$ con precisión arbitraria independientemente del principio de Heisenberg.

En la figura 2.1 ilustramos un neutrón puntual que a un cierto tiempo t está ubicado en una posición espacial $\mathbf{x}$ y se mueve en línea recta en una dirección $\hat{\mathbf{\Omega}}$ con una energía $E = 1/2 \cdot mv^2$.

2.1. Secciones eficaces

Definición 2.1 (sección eficaz macroscópica total). La *sección eficaz macroscópica total* Σ_t de un medio es tal que el producto

$$\Sigma_t \cdot ds$$

es la probabilidad de que un neutrón tenga una colisión con el núcleo de algún átomo del material por el que viaja a lo largo de una distancia ds en línea recta. Es decir, la sección eficaz macroscópica es el número de colisiones esperadas por neutrón y por unidad de longitud lineal. Sus unidades son inversa de longitud, es decir m^{-1} o cm^{-1}.

Además de referirnos a la sección eficaz[3] total, podemos particularizar el concepto al tipo de reacción k, es decir, $\Sigma_k \cdot ds$ es la probabilidad de que un neutrón tenga una reacción de tipo k en el

[1] *Pun intended.*
[2] Llegado el caso veremos cómo reducir el problema para casos particulares de dominios de una y dos dimensiones.
[3] En inglés es *cross section.*

intervalo espacial de longitud ds. En nuestro caso particular, la reacción genérica k puede ser particularizada según el subíndice a

Subíndice	Reacción
t	total
c	captura radiativa
f	fisión
a	absorción ($\Sigma_c + \Sigma_f$)
s	dispersión (scattering)

Las secciones eficaces macroscópicas dependen de la energía E del neutrón incidente y de las propiedades del medio que provee los núcleos blanco. Como éstas dependen del espacio (usualmente a través de otras propiedades intermedias como por ejemplo temperaturas, densidades o concentraciones de impurezas), en general las secciones eficaces macroscópicas son funciones tanto del espacio $\mathbf{x}$ como de la energía E, es decir $\Sigma_k = \Sigma_k(\mathbf{x}, E)$. En este trabajo no consideramos una dependencia explícita con el tiempo t.

Una forma de incorporar el concepto de sección eficaz macroscópica es pensar que ésta proviene del producto de una sección eficaz microscópica σ_k (con unidades de área) y una densidad atómica n (con unidades de inversa de volumen) del medio

$$\Sigma_k[\mathrm{cm}^{-1}] = \sigma_k[\mathrm{cm}^2] \cdot n[\mathrm{cm}^{-3}]$$

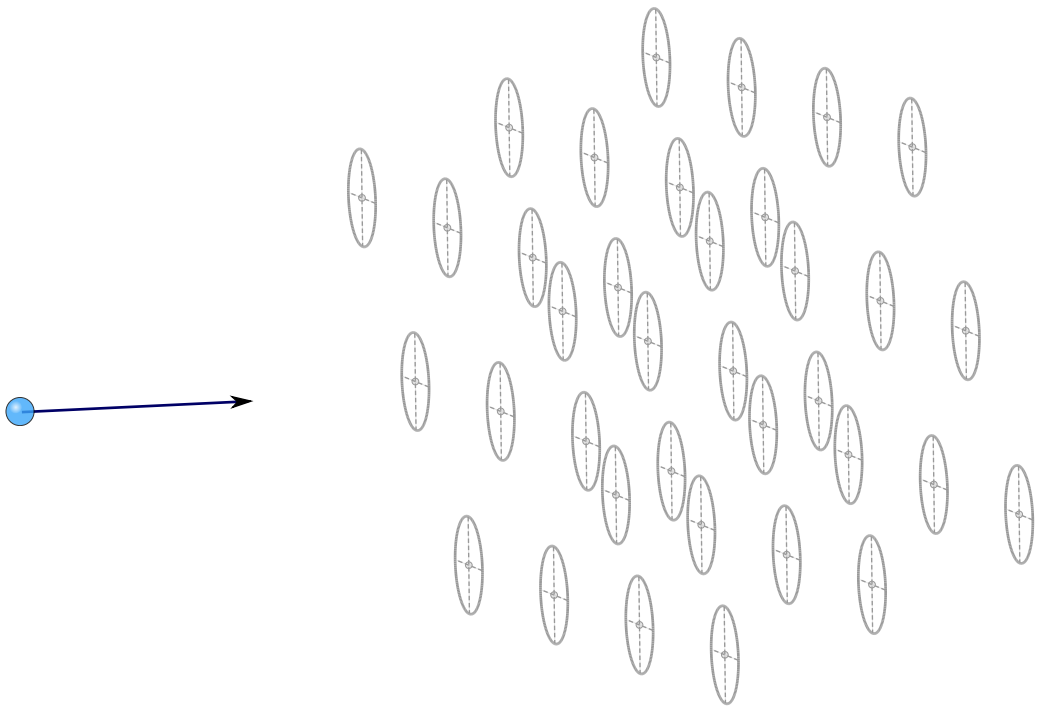


Figura 2.2.: Interpretación de la sección eficaz microscópica como el área asociada a un núcleo transversal a la dirección de viaje del neutrón incidente.

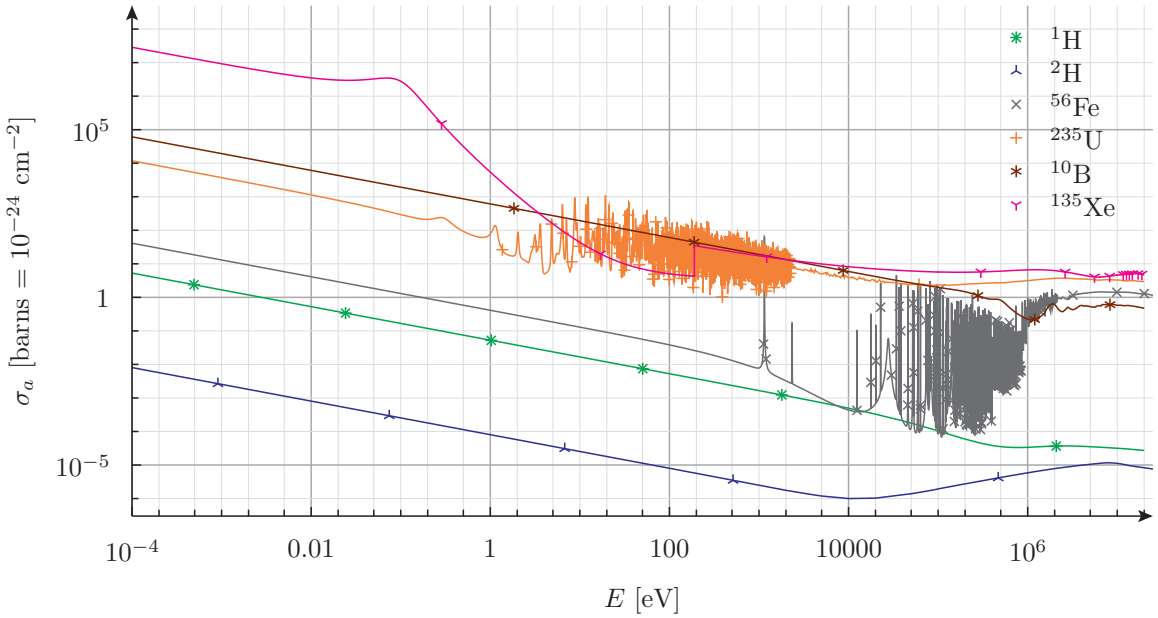


Figura 2.3.: Dependencia de la sección eficaz microscópica de absorción σ_a con respecto a la energía E del neutrón incidente para diferentes isótopos blanco.

La sección eficaz microscópica σ_k tiene efectivamente unidades de área (típicamente del orden de 10^{-24} cm^2, unidad que llamamos *barn*[4]) y se interpreta como el área asociada a un núcleo transversal a la dirección de viaje de un neutrón tal que si este neutrón pasara a través de dicha área, se llevaría a cabo una reacción de tipo k (figura 2.2). Las secciones eficaces microscópicas dependen no solamente de las propiedades nucleares de los núcleos blanco sino que también dependen fuertemente de la energía E del neutrón incidente, llegando a cambiar varios órdenes de magnitud debido a efectos de resonancias como podemos observar en la figura 2.3. Además, σ_k depende de la temperatura T del medio que define la forma en la cual los átomos se mueven por agitación térmica alrededor de su posición de equilibrio ya que se produce un efecto tipo Doppler entre el neutrón y el núcleo blanco que modifica la sección eficaz microscópica [79], [80]. Por lo tanto, para un cierto isótopo, $\sigma_k = \sigma_k(E, T)$.

Por otro lado, la densidad atómica n del medio depende de la densidad termodinámica ρ, que a su vez depende de su estado termodinámico usualmente definido por la presión p y la temperatura T. Como estas variables pueden depender de forma arbitraria del espacio $\mathbf{x}$, podemos escribir efectivamente

$$\Sigma_k = n \cdot \sigma_k = n\Big(p(\mathbf{x}), T(\mathbf{x})\Big) \cdot \sigma_k\Big(E, T(\mathbf{x})\Big) = \Sigma_k(\mathbf{x}, E)$$

Las ideas presentadas son válidas para un único isótopo libre de cualquier influencia externa. En los reactores nucleares reales, por un lado existen efectos no lineales como por ejemplo el hecho de los átomos de hidrógeno o deuterio y los de oxígeno no están libres en la molécula de agua. Esto hace que las secciones eficaces del todo (i.e. de un conjunto de átomos enlazados covalentemente) no sean iguales a la suma algebraica de las partes y debamos calcular las secciones eficaces macroscópicas

[4]Se dice que durante las primeras mediciones experimentales de secciones eficaces los físicos americanos esperaban encontrar resultados del orden de las áreas transversales asociadas a los tamaños geométricos de los núcleos. Pero encontraron valores mucho más grandes, por lo que decían a modo de broma "this cross section is as big as a barn."

con una metodología más apropiada (ver sección 2.5.1). Por otro lado, justamente en los reactores nucleares las reacciones que interesan son las que dan como resultado la transmutación de materiales por lo que continuamente la densidad atómica n de todos los isótopos varía con el tiempo. En este trabajo, no vamos a tratar con la dependencia de las secciones eficaces con el tiempo explícitamente sino que llegado el caso, como discutimos en la sección 2.5, daremos la dependencia implícitamente a través de otras propiedades intermedias tales como la evolución del quemado del combustible y/o la concentración de xenón 135 en las pastillas de de dióxido de uranio en forma cuasi-estática.

Observación. Si bien la sección eficaz de un material que es combinación de otros no es la combinación lineal de los componentes, es cierto que las sumas de secciones eficaces correspondientes a reacciones parciales dan como resultado la sección eficaz de la reacción total. Por ejemplo, $\Sigma_a = \Sigma_c + \Sigma_f$ y $\Sigma_t = \Sigma_a + \Sigma_s$.

A partir de este momento suponemos que conocemos las secciones eficaces macroscópicas en función del vector posición $\mathbf{x}$ y de la energía E para todos los problemas que planteamos, y que además éstas no dependen del tiempo t.

2.1.1. Dispersión de neutrones

Cuando un neutrón que viaja en una cierta dirección $\hat{\mathbf{\Omega}}$ con una energía E colisiona con un núcleo blanco en una reacción de dispersión o *scattering*,[5] tanto el neutrón como el núcleo blanco intercambian energía. En este caso podemos pensar que luego de la colisión, el neutrón incidente se ha transformado en otro neutrón emitido en una nueva dirección $\hat{\mathbf{\Omega}}'$ con una nueva energía E'. Para tener este efecto en cuenta, utilizamos el concepto que sigue.

Definición 2.2 (sección eficaz de scattering diferencial). La *sección eficaz de scattering diferencial* Σ_s tal que

$$\Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E')\, d\hat{\mathbf{\Omega}}'\, dE'$$

es la probabilidad por unidad de longitud lineal que un neutrón de energía E viajando en la dirección $\hat{\mathbf{\Omega}}$ sea dispersado hacia un intervalo de energía entre E' y $E' + dE'$ y a un cono $d\hat{\mathbf{\Omega}}'$ alrededor de la dirección $\hat{\mathbf{\Omega}}'$. Sus unidades son inversa de longitud por inversa de ángulo sólido por inversa de energía.

Utilizando argumentos de simetría, podemos mostrar que la sección eficaz diferencial de scattering Σ_s sólo puede depender del producto interno $\mu = \hat{\mathbf{\Omega}} \cdot \hat{\mathbf{\Omega}}'$ y no separadamente de $\hat{\mathbf{\Omega}}$ y de $\hat{\mathbf{\Omega}}'$. En la figura 2.4 ilustramos la idea.

En general podemos separar a la sección eficaz diferencial (definición 2.2) en una sección eficaz total Σ_{s_t} y en una probabilidad de distribución angular y energética ξ_s tal que

$$\Sigma_s(\mathbf{x}, \mu, E \to E') = \Sigma_{s_t}(\mathbf{x}, E) \cdot \xi_s(\mu, E \to E') \tag{2.1}$$

[5] Como ya hemos dicho, el término español "dispersión" como traducción del concepto de "scattering" no es muy feliz. A partir este punto, durante el resto de esta tesis usamos solamente la palabra *scattering* para referirnos a este concepto.

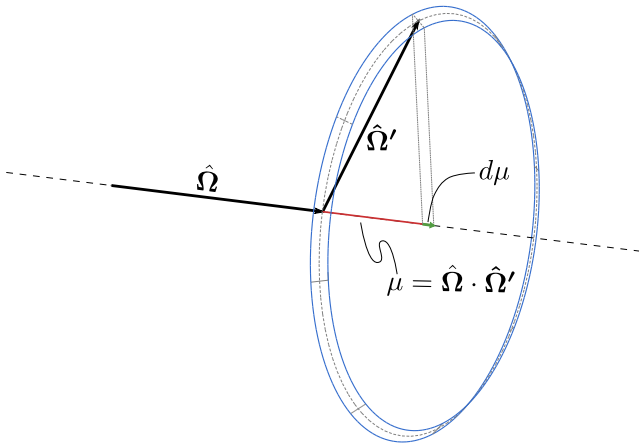


Figura 2.4.: Debido a la simetría azimutal, el scattering no depende de las direcciones $\hat{\mathbf{\Omega}}$ y de $\hat{\mathbf{\Omega}}'$ en forma separada sino que depende del coseno del ángulo entre ellas $\mu = \hat{\mathbf{\Omega}} \cdot \hat{\mathbf{\Omega}}'$.

donde Σ_{s_t} es la sección eficaz macroscópica *total* de scattering, que da la probabilidad por unidad de longitud de que un neutrón incidente de energía E inicie un proceso de scattering y la función ξ_s describe la distribución de neutrones emergentes.

Podemos integrar ambos miembros de la ecuación 2.1 con respecto a E' y a μ, y despejar Σ_{s_t} para obtener su definición

$$\Sigma_{s_t}(\mathbf{x}, E) = \frac{\displaystyle\int_0^{\infty}\int_{-1}^{+1} \Sigma_s(\mathbf{x}, \mu, E \to E')\, d\mu\, dE'}{\displaystyle\int_0^{\infty}\int_{-1}^{+1} \xi_s(\mu, E \to E')\, d\mu\, dE'}$$

El denominador tiene que ser igual a uno ya que en la reacción de scattering el neutrón dispersado tiene que tener alguna dirección μ y alguna energía E'. Luego

$$\Sigma_{s_t}(\mathbf{x}, E) = \int_0^{\infty}\int_{-1}^{+1} \Sigma_s(\mathbf{x}, \mu, E \to E')\, d\mu\, dE' \tag{2.2}$$

2.1.2. Expansión en polinomios de Legendre

Teorema 2.1 (expansión en polinomios de Legendre). *Cualquier función $f(\mu) : \mu \in [-1, +1] \to \mathbb{R}^1$ de cuadrado integrable puede ser escrita como la suma infinita de un coeficiente f_ℓ por el* polinomio de Legendre $P_\ell(\mu)$ *de grado $\ell \geq 0$*

$$f(\mu) = \sum_{\ell=0}^{\infty} f_\ell \cdot P_\ell(\mu)$$

con la condición de normalización

$$P_\ell(1) = 1$$

Definición 2.3. Los primeros polinomios de Legendre (figura 2.5) son

$$\begin{aligned}
P_0(\mu) &= 1 \\
P_1(\mu) &= \mu \\
P_2(\mu) &= \frac{1}{2}\left(3\mu^2 - 1\right) \\
P_3(\mu) &= \frac{1}{2}\left(5\mu^3 - 3\mu\right) \\
P_4(\mu) &= \frac{1}{8}\left(35\mu^4 - 30\mu^2 + 3\right) \\
P_5(\mu) &= \frac{1}{8}\left(63\mu^5 - 70\mu^3 + 15\mu\right) \\
P_6(\mu) &= \frac{1}{16}\left(231\mu^6 - 315\mu^4 + 105\mu^2 - 5\right)
\end{aligned}$$

Definición 2.4 (la delta de Kronecker).

$$\delta_{\ell\ell'} = \begin{cases} 1 & \text{si } \ell = \ell' \\ 0 & \text{si } \ell \neq \ell' \end{cases}$$

Teorema 2.2 (ortogonalidad de los polinomios de Legendre). *Los polinomios de Legendre son ortogonales. Más aún,*

$$\int_{-1}^{+1} P_\ell(\mu) \cdot P_{\ell'}(\mu)\, d\mu = \frac{2}{2\ell + 1} \cdot \delta_{\ell\ell'}$$

Corolario 2.1. *Los coeficientes f_ℓ de la expansión de $f(\mu)$ en polinomios de Legendre del teorema 2.1 son iguales a*

$$f_\ell = \frac{2\ell + 1}{2} \cdot \int_{-1}^{1} f(\mu) \cdot P_\ell(\mu)\, d\mu$$

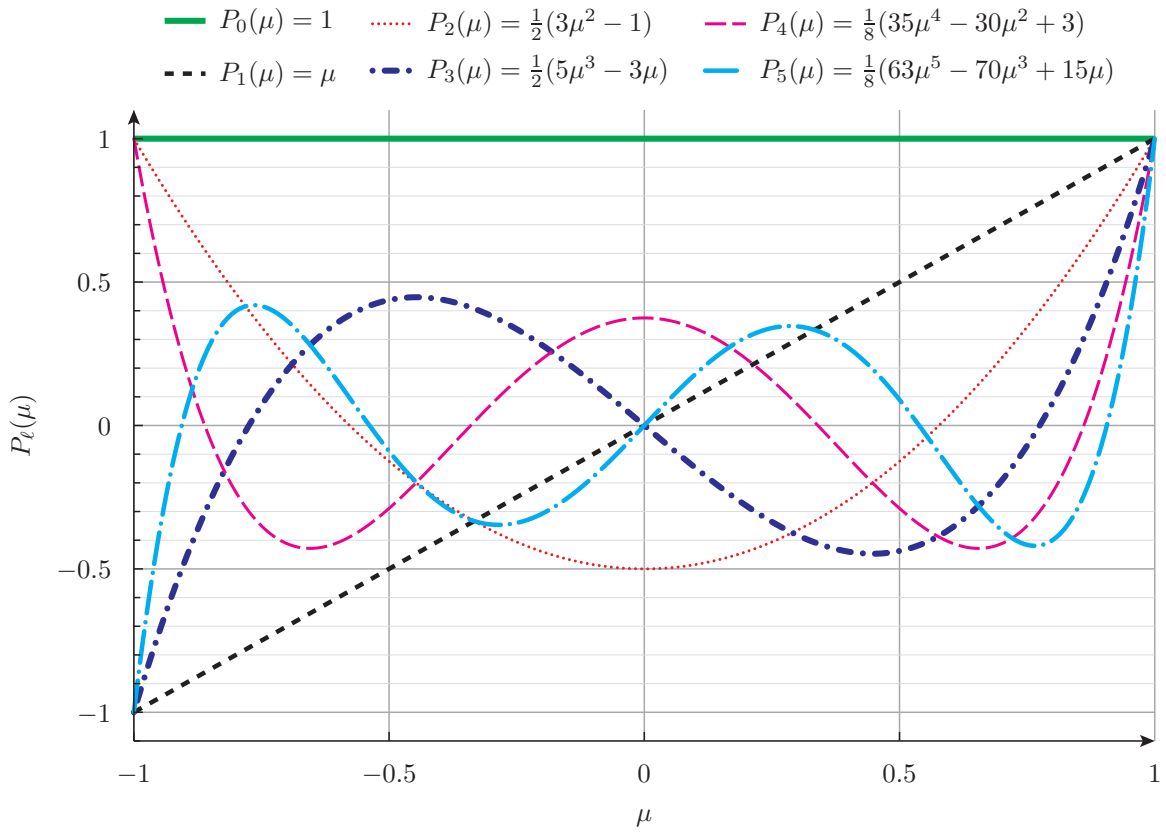


Figura 2.5.: Primeros seis polinomios de Legendre $P_\ell(\mu)$, $\ell = 0, \ldots, 5$.

Una forma de tener en cuenta la dependencia de Σ_s con μ en la ecuación 2.1 es recurrir a una expansión en polinomios de Legendre. En efecto, para una cierta posición $\mathbf{x}$ y dos energías E y E' fijas, la sección eficaz Σ_s de la ecuación 2.1 depende de un único escalar $-1 \leq \mu \leq 1$ sin presentar singularidades, es decir es una función de cuadrado integrable. Entonces podemos escribir[6]

$$\Sigma_s(\mathbf{x}, \mu, E \to E') = \sum_{\ell=0}^{\infty} \frac{2\ell + 1}{2} \Sigma_{s_\ell}(\mathbf{x}, E \to E') \cdot P_\ell(\mu) \tag{2.3}$$

donde los coeficientes son

$$\Sigma_{s_\ell}(\mathbf{x}, E \to E') = \int_{-1}^{+1} \Sigma_s(\mathbf{x}, \mu, E \to E') \cdot P_\ell(\mu)\, d\mu \tag{2.4}$$

Corolario 2.2. *La sección eficaz de scattering total Σ_{s_t} sólo depende de Σ_{s_0}.*

[6]El coeficiente $(2\ell + 1)/2$ aparece para que las expresiones que siguen sean consistentes con los usos y costumbres históricos de la evaluación de secciones eficaces (sección 2.5.1) y de códigos de celda (sección 2.5.2). En particular, aparece para que en la ecuación 2.5 ambos miembros tengan coeficientes unitarios. Es posible dar otras definiciones y desarrollar consistentemente la matemática para llegar a expresiones finales igualmente válidas, pero ello modificaría la definición de los coeficientes de la expansión dados por el corolario 2.1 y haría que las secciones eficaces calculadas a nivel de celda no puedan ser introducidas directamente en la entrada del código de núcleo que describimos en el capítulo 4. En particular, arribar a la ecuación 2.8 es de interés para la consistencia de las secciones eficaces entre códigos de diferente nivel. La referencia [34] utiliza otra forma de expandir el kernel de scattering que resulta en un factor dos de diferencia con respecto a la ecuación 2.4.

Demostración. Reemplazando la expansión dada por la ecuación 2.3 en la ecuación 2.2 tenemos

$$\begin{aligned}\Sigma_{s_t}(\mathbf{x}, E) &= \int_0^\infty \int_{-1}^{+1} \sum_{\ell=0}^{\infty} \frac{2\ell+1}{2} \Sigma_{s_\ell}(\mathbf{x}, E \to E') \cdot P_\ell(\mu)\, d\mu\, dE' \\ &= \int_0^\infty \left[\sum_{\ell=0}^{\infty} \frac{2\ell+1}{2} \Sigma_{s_\ell}(\mathbf{x}, E \to E') \cdot \int_{-1}^{+1} P_\ell(\mu)\, d\mu \right] dE'\end{aligned}$$

Según el teorema 2.2, todos los polinomios de Legendre de orden $\ell \geq 1$ son ortogonales con respecto a $P_0(\mu) = 1$, por lo que

$$\int_{-1}^{+1} P_\ell(\mu)\, d\mu \begin{cases} 2 & \text{para } \ell = 0 \\ 0 & \text{para } \ell > 0 \end{cases}$$

Luego la única contribución a la suma infinita diferente de cero es la correspondiente a $\ell = 0$

$$\Sigma_{s_t}(\mathbf{x}, E) = \int_0^\infty \left[\frac{2 \cdot 0 + 1}{2} \cdot \Sigma_{s_0}(\mathbf{x}, E \to E') \cdot 2 \right] dE' = \int_0^\infty \Sigma_{s_0}(\mathbf{x}, E \to E') dE' \tag{2.5}$$

□

Recordando que $\mu = \hat{\mathbf{\Omega}} \cdot \hat{\mathbf{\Omega}}'$, debe cumplirse que

$$\int_{4\pi} \Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E')\, d\hat{\mathbf{\Omega}}' = \int_{-1}^{+1} \Sigma_s(\mathbf{x}, \mu, E \to E')\, d\mu \tag{2.6}$$

Si tenemos scattering isótropo en el marco de referencia del laboratorio,[7] entonces

a. el integrando del miembro izquierdo de la ecuación 2.6 no depende de $\hat{\mathbf{\Omega}}'$ y puede salir fuera de la integral, y
b. $\Sigma_s(\mathbf{x}, \mu, E \to E')$ no depende de μ y el único coeficiente Σ_{s_ℓ} diferente de cero es el correspondiente a $\ell = 0$ con lo que

$$\int_{-1}^{+1} \Sigma_s(\mathbf{x}, \mu, E \to E')\, d\mu = \int_{-1}^{+1} \frac{2 \cdot 0 + 1}{2} \cdot \Sigma_{s_0}(\mathbf{x}, E \to E')\, d\mu = \Sigma_{s_0}(\mathbf{x}, E \to E') \tag{2.7}$$

Reemplazando la ecuación 2.7 en la 2.6 y sacando $\Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E')$ fuera de la integral tenemos

$$\Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E') \cdot 4\pi = \Sigma_{s_0}(\mathbf{x}, E \to E')$$

[7] Como ya dijimos, esta nomenclatura es puramente académica. Una expresión más apropiada según la potencial aplicación industrial de los conceptos desarrollados en esta tesis sería "marco de referencia de *la central nuclear*".

con lo que la sección eficaz diferencial (definición 2.2) para scattering isótropo en el sistema del reactor es igual al coeficiente Σ_{s_0} dividido 4π

$$\Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E') = \frac{1}{4\pi} \cdot \Sigma_{s_0}(\mathbf{x}, E \to E') \tag{2.8}$$

Observación. Según el punto b y la ecuación 2.3,

$$\begin{aligned}\Sigma_s(\mathbf{x}, \mu, E \to E') &= \frac{2 \cdot 0 + 1}{2} \cdot \Sigma_{s_0}(\mathbf{x}, E \to E') \cdot P_0(\mu) \\ &= \frac{1}{2} \cdot \Sigma_{s_0}(\mathbf{x}, E \to E')\end{aligned}$$

Comparando este resultado con la ecuación 2.8 llegamos a la conclusión que

$$\Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E') = \frac{1}{2\pi} \cdot \Sigma_s(\mathbf{x}, \mu, E \to E') \tag{2.9}$$

Este resultado se explica ya que dado un ángulo $\hat{\mathbf{\Omega}}$ fijo, para cada posible valor de μ hay 2π posibles valores de $\hat{\mathbf{\Omega}}'$ que pueden dar como resultado el mismo coseno.

Si en cambio el scattering resulta ser completamente elástico e isótropo pero en el marco de referencia del centro de masa del sistema compuesto por el neutrón incidente y el núcleo blanco, entonces a cada energía de salida E' le corresponde un único ángulo de scattering μ a través de las leyes clásicas de conservación de energía y momento lineal. Siguiendo el desarrollo de la referencia [58], la sección eficaz diferencial es

$$\Sigma_s(\mathbf{x}, \mu, E \to E') = \begin{cases} \Sigma_{s_t}(\mathbf{x}, E) \cdot \dfrac{\delta(\mu - \mu_0)}{(1-\alpha)E} & \text{para } \alpha E < E' < E \\ 0 & \text{de otra manera} \end{cases}$$

donde ahora $\delta(x)$ es la distribución delta de Dirac (no confundir con la δ de Kronecker de la definición 2.4) y

$$\alpha(A) = \frac{(A-1)^2}{(A+1)^2}$$

$$\mu_0(A, E, E') = \frac{1}{2}\left[(A+1)\sqrt{\frac{E'}{E}} - (A-1)\sqrt{\frac{E}{E'}}\right] \tag{2.10}$$

siendo A es el número de masa del núcleo blanco. Llamamos a la magnitud μ_0 coseno medio de scattering.

Observación. Esta nomenclatura para μ_0 es general pero la expresión matemática dada por la ecuación 2.10 es particular para el caso de scattering elástico e isótropo en el marco de referencia del centro de masa. En la ecuación 2.12 generalizamos la definición para cualquier tipo de scattering.

La expresión para el ℓ-ésimo coeficiente de la expansión en polinomios de Legendre para $\alpha E < E' < E$ según la ecuación 2.4 es

$$\Sigma_{s_\ell}(\mathbf{x}, E \to E') = \frac{\Sigma_{s_t}(\mathbf{x}, E)}{(1-\alpha)E} \int_{-1}^{+1} \delta(\mu - \mu_0) \cdot P_\ell(\mu)\, d\mu = \frac{\Sigma_{s_t}(\mathbf{x}, E)}{(1-\alpha)E} \cdot P_\ell(\mu_0)$$

por lo que

$$\Sigma_s(\mathbf{x}, \mu, E \to E') = \frac{\Sigma_{s_t}(\mathbf{x}, E)}{(1-\alpha)E} \cdot \sum_{\ell=0}^{\infty} \frac{2\ell+1}{2} \cdot P_\ell(\mu_0) \cdot P_\ell(\mu)$$

Tomando solamente los dos primeros términos y recordando que $P_1(\mu) = \mu$, podemos aproximar

$$\Sigma_s(\mathbf{x}, \mu, E \to E') \approx \begin{cases} \frac{\Sigma_{s_t}(\mathbf{x},E)}{(1-\alpha)E} \cdot \left(1 + \frac{3 \cdot \mu_0 \cdot \mu}{2}\right) & \text{para } \alpha E < E' < E \\ 0 & \text{para } E' < \alpha E \end{cases}$$

Estas dos ideas nos permiten introducir los siguientes conceptos.

Definición 2.5 (scattering isótropo). Decimos que hay *scattering isótropo* (a partir de ahora siempre nos vamos a referir al marco de referencia del reactor) cuando los coeficientes de la expansión de la sección eficaz diferencial de scattering $\Sigma_s(\mathbf{x}, \mu, E \to E')$ en polinomios de Legendre son todos nulos excepto el correspondiente a $\ell = 0$. En este caso, la sección eficaz diferencial no depende del ángulo y vale la ecuación 2.8:

$$\Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E') = \frac{1}{4\pi} \cdot \Sigma_{s_0}(\mathbf{x}, E \to E') \tag{2.8}$$

Definición 2.6 (scattering linealmente anisótropo). Si además de Σ_{s_0} resulta que el único otro coeficiente diferente de cero es Σ_{s_1} correspondiente a $\ell = 1$ entonces decimos que el scattering es *linealmente anisótropo*, y la sección eficaz diferencial es la suma de la sección eficaz total más un coeficiente multiplicado por el coseno del ángulo de scattering:

$$\begin{aligned} \Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E') &= \frac{1}{4\pi} \cdot \left[\Sigma_{s_0}(\mathbf{x}, E \to E') + (2 \cdot 1 + 1) \cdot \Sigma_{s_1}(\mathbf{x}, E \to E') \cdot P\left(\hat{\mathbf{\Omega}} \cdot \hat{\mathbf{\Omega}}'\right)\right] \\ &= \frac{1}{4\pi} \cdot \left[\Sigma_{s_0}(\mathbf{x}, E \to E') + 3 \cdot \Sigma_{s_1}(\mathbf{x}, E \to E') \cdot \left(\hat{\mathbf{\Omega}} \cdot \hat{\mathbf{\Omega}}'\right)\right] \end{aligned} \tag{2.11}$$

Definición 2.7 (coseno medio). Definimos el *coseno medio del ángulo de scattering* μ_0 para una ley de dispersión general como

$$\mu_0(\mathbf{x}, E) = \frac{\displaystyle\int_0^\infty \int_{-1}^{+1} \mu \cdot \Sigma_s(\mathbf{x}, \mu, E \to E')\, d\mu\, dE'}{\displaystyle\int_0^\infty \int_{-1}^{+1} \Sigma_s(\mathbf{x}, \mu, E \to E')\, d\mu\, dE'} = \frac{\displaystyle\int_0^\infty \Sigma_{s_1}(\mathbf{x}, E \to E')\, dE'}{\displaystyle\int_0^\infty \Sigma_{s_0}(\mathbf{x}, E \to E')\, dE'} \tag{2.12}$$

En el caso de scattering general, i.e. no necesariamente isótropo en algún marco de referencia y no necesariamente elástico, debemos conocer entonces

a. la dependencia explícita de Σ_s con $\hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}'$ (que puede ser aproximada mediante evaluaciones discretas), o
b. una cierta cantidad de coeficientes Σ_{s_ℓ} de su desarrollo en polinomios de Legendre sobre μ.

En esta tesis trabajamos a lo más con scattering linealmente anisótropo. Esto es, suponemos que la sección eficaz diferencial de scattering está dada por la ecuación 2.11 y suponemos que conocemos tanto Σ_{s_0} como Σ_{s_1} en función del espacio y de los grupos de energías discretizados antes de resolver la ecuación de transporte a nivel de núcleo (ver sección 2.5).

2.1.3. Fisión de neutrones

Cuando un núcleo pesado se fisiona en dos núcleos más pequeños, ya sea debido a una fisión espontánea o a una fisión inducida por la absorción de un neutrón, se liberan además de los productos de fisión propiamente dichos y radiación γ debida al re-acomodamiento de los niveles energéticos de los nucleones que intervienen en la reacción, entre dos y tres neutrones. Llamamos $\nu(\mathbf{x}, E)$ a la cantidad promedio de neutrones liberados por cada fisión. El valor numérico de $2 < \nu < 3$ depende de la energía E del neutrón incidente y de la composición del material combustible el punto $\mathbf{x}$.

Tal como hicimos en la sección sección 2.1.1, separamos la probabilidad Σ_f de que un neutrón incidente de energía E produzca una fisión unidad de longitud de viaje colisionando con un núcleo blanco en la posición $\mathbf{x}$ en una sección eficaz total Σ_{f_t} y una distribución ξ_f en ángulo y energía cada uno de los neutrones nacidos por la fisión

$$\Sigma_f(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E') = \Sigma_{f_t}(\mathbf{x}, E) \cdot \xi_f(\hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E')$$

La distribución en energía de los neutrones nacidos por fisión está dada por el espectro de fisión χ, que definimos a continuación.

Definición 2.8 (espectro de fisión). El *espectro de fisión* $\chi(E)$ es tal que

$$\chi(E)\, dE$$

es la probabilidad de que un neutrón nacido en una fisión lo haga con una energía dentro del intervalo $[E, E + dE]$. El espectro de fisión está normalizado de forma tal que

$$\int_0^\infty \chi(E)\, dE = 1 \tag{2.13}$$

Experimentalmente se encuentra que los neutrones de fisión nacen isotrópicamente en el marco de referencia del reactor independientemente de la energía E del neutrón incidente que la provocó. Además, la energía E' con la que emergen tampoco depende de la energía del E neutrón incidente. Luego ξ_f no depende ni de $\hat{\mathbf{\Omega}}$ ni de $\hat{\mathbf{\Omega}}'$ y podemos separarla en una cierta dependencia de $\Xi(E)$ que da la probabilidad de generar una fisión, multiplicada por el espectro de fisión $\chi(E')$:

$$\xi_f(\hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E') = \Xi(E) \cdot \chi(E')$$

Como cada neutrón nacido por fisión tiene que tener alguna dirección de vuelo $\hat{\mathbf{\Omega}}'$ y alguna energía E', entonces la integral de ξ_f sobre todas las posibles energías E' y ángulos $\hat{\mathbf{\Omega}}'$ debe ser igual a uno. Entonces

$$\int_0^\infty \int_{4\pi} \xi_f(\hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E')\, d\hat{\mathbf{\Omega}}'\, dE' = \Xi(E) \cdot \int_0^\infty \int_{4\pi} \chi(E')\, d\hat{\mathbf{\Omega}}'\, dE' = 1$$

Teniendo en cuenta la normalización de χ dada por la ecuación 2.13, resulta

$$\Xi(E) = \frac{1}{4\pi}$$

por lo que

$$\Sigma_f(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E') = \frac{\chi(E')}{4\pi} \cdot \Sigma_{f_t}(\mathbf{x}, E) \tag{2.14}$$

Observación. Dado que en la página 30 hemos supuesto que la población neutrónica es igual al valor medio de la distribución naturalmente estocástica de los neutrones, desde el punto de vista del desarrollo matemático cada neutrón que produce una fisión genera exactamente $\nu(\mathbf{x}, E)$ neutrones de fisión. Como la dependencia de ν es la misma que la de sección eficaz total de fisión Σ_{f_t} entonces englobamos el producto $\nu \cdot \Sigma_f$ como una única función $\nu\Sigma_f(\mathbf{x}, E)$.

Durante la operación de un reactor, no todos los neutrones provenientes de la fisión aparecen en el mismo instante en el que se produce. Una cierta fracción β de todos los neutrones son producto del decaimiento radioactivo de

a. los productos de fisión, o
b. de los hijos de los productos de fisión.

En cualquier caso, en cálculos transitorios es necesario distinguir entre la fracción $1-\beta$ de neutrones instantáneos[8] que aparecen en el mismo momento de la fisión y la fracción β de neutrones retardados que aparecen más adelante. Para ello dividimos a los neutrones retardados en I grupos, les asignamos una fracción β_i y una constante de tiempo λ_i, para $i = 1, \ldots, N$ y definimos un mecanismo de aparición exponencial para cada uno de ellos.

Como discutimos más adelante en la sección 3.5, en cálculos estacionarios no es necesario realizar esta división entre neutrones instantáneos y retardados ya que eventualmente todos los neutrones estarán contribuyendo a la reactividad neta del reactor. En el caso particular en el que no haya una fuente externa de neutrones sino que todas las fuentes se deban a fisiones la probabilidad de que el reactor esté exactamente crítico es cero. Para poder realizar cálculos estacionarios y además tener una idea de la distancia a la criticidad debemos recurrir a un reactor crítico asociado, cuya forma más usual es el *reactor crítico asociado en k* introducido más adelante en la definición 3.27. En este

[8] En el sentido del inglés *prompt*.

caso, dividimos arbitrariamente las fuentes de fisión por un número real $k_{\text{eff}} \sim 1$ que pasa a ser una incógnita del problema y cuya diferencia relativa con respecto a la unidad da una idea de la distancia a la criticidad del reactor original.

2.2. Flujos y ritmos de reacción

El problema central del cálculo de reactores es la determinación de la distribución espacial y temporal de los neutrones dentro del núcleo de un reactor nuclear. En esta sección desarrollamos la matemática para el caso de $\mathbf{x} \in \mathbb{R}^3$. En casos particulares aclaramos cómo debemos proceder para problemas en una y en dos dimensiones.

Definición 2.9 (densidad de neutrones). La *distribución de densidad de neutrones* N en un espacio de las fases de siete dimensiones $\mathbf{x} \in \mathbb{R}^3$, $\hat{\mathbf{\Omega}} \in \mathbb{R}^2$,[9] $E \in \mathbb{R}$ y $t \in \mathbb{R}$ tal que

$$N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d^3\mathbf{x}\, d\hat{\mathbf{\Omega}}\, dE$$

es el número de neutrones (en el sentido de la media estadística dada la naturaleza probabilística del comportamiento de los neutrones) en un elemento volumétrico $d^3\mathbf{x}$ ubicado alrededor del punto $\mathbf{x}$ del espacio viajando en el cono de direcciones de magnitud $d\hat{\mathbf{\Omega}}$ alrededor de la dirección $\hat{\mathbf{\Omega}}$ con energías entre E y $E + dE$ en el tiempo t.[10]

Definición 2.10 (flujo angular). El *flujo angular* ψ es el producto entre la velocidad v y la distribución de densidad N de los neutrones

$$\begin{aligned} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) &= v(E) \cdot N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \\ &= \sqrt{\frac{2E}{m}} \cdot N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \end{aligned} \tag{2.15}$$

donde $v(E)$ es la velocidad clásica correspondiente a un neutrón de masa m cuya energía cinética es $E = 1/2 \cdot mv^2$.

Esta magnitud es más útil para evaluar ritmos de colisiones y reacciones que la densidad de neutrones N.

Corolario 2.3. *Como $v \cdot dt$ es la distancia lineal ds que viaja un neutrón de velocidad v, entonces*

$$\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d^3\mathbf{x}\, d\hat{\mathbf{\Omega}}\, dE\, dt = v(E) \cdot N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, dt \;\; d^3\mathbf{x}\, d\hat{\mathbf{\Omega}}\, dE$$

es el número total de longitudes lineales que los neutrones han viajado sobre un cono $d\hat{\mathbf{\Omega}}$ alrededor de la dirección $\hat{\mathbf{\Omega}}$ con energía en el intervalo $[E, E + dE]$ que estaban en un intervalo de tiempo $[t, t + dt]$ en un diferencial de volumen $d^3\mathbf{x}$ en la posición $\mathbf{x}$.

[9] Si bien la dirección $\hat{\mathbf{\Omega}} = [\Omega_x\ \Omega_y\ \Omega_z]^T$ tiene tres componentes llamados *cosenos dirección*, sólo dos son independientes (por ejemplo las coordenadas angulares cenital θ y azimutal φ) ya que debe cumplirse que $\Omega_x^2 + \Omega_y^2 + \Omega_z^2 = 1$.

[10] Como ya hemos mencionado, no tenemos en cuenta el principio de Heisenberg.

Corolario 2.4. *La probabilidad de que un neutrón tenga una reacción de tipo k durante la distancia lineal de vuelo ds, según la definición 2.1, es $\Sigma_k \cdot ds$. Entonces la expresión*

$$\Sigma_k(\mathbf{x}, E) \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d^3\mathbf{x}\, d\hat{\mathbf{\Omega}}\, dE\, dt$$

es el número de reacciones de tipo k en el diferencial de volumen de fases $d^3\mathbf{x}\, d\hat{\mathbf{\Omega}}\, dE\, dt$.

Para obtener el número total de reacciones de todos los neutrones independientemente de la dirección $\hat{\mathbf{\Omega}}$ del neutrón incidente debemos integrar esta cantidad sobre todos los posibles ángulos de incidencia. Para ello utilizamos el concepto que sigue.

Definición 2.11 (flujo escalar). El *flujo escalar* ϕ es la integral del flujo angular sobre todas las posibles direcciones de viaje de los neutrones:

$$\phi(\mathbf{x}, E, t) = \int_{4\pi} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d\hat{\mathbf{\Omega}} \tag{2.16}$$

Corolario 2.5. *Con estas definiciones, el ritmo R_k de reacciones de tipo k por unidad de tiempo en $d^3\mathbf{x}\, dE$ es*

$$R_k(\mathbf{x}, E, t)\, d^3\mathbf{x}\, dE = \Sigma_k(\mathbf{x}, E) \cdot \phi(\mathbf{x}, E, t)\, d^3\mathbf{x}\, dE \tag{2.17}$$

con lo que el producto $R_t = \Sigma_t \phi$ da una expresión simple para la distribución del ritmo de reacciones totales por unidad de volumen y de energía.

Definición 2.12 (corriente). El *vector corriente* $\mathbf{J}$ es la integral del producto entre el flujo angular y el versor de dirección de viaje de los neutrones $\hat{\mathbf{\Omega}}$ sobre todas las direcciones de viaje:

$$\mathbf{J}(\mathbf{x}, E, t) = \int_{4\pi} \left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \cdot \hat{\mathbf{\Omega}}\right]\, d\hat{\mathbf{\Omega}}$$

Observación. La corriente es una cantidad vectorial ya que el integrando es un vector cuya magnitud es igual al flujo angular y cuya dirección es la del versor $\hat{\mathbf{\Omega}}$ sobre el cual estamos integrando.

Observación. El producto escalar entre el vector corriente $\mathbf{J}$ y un cierto versor espacial $\hat{\mathbf{n}}$

$$J_n(\mathbf{x}, E, t) = \hat{\mathbf{n}} \cdot \mathbf{J}(\mathbf{x}, E, t) = \int_{4\pi} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \cdot \left(\hat{\mathbf{\Omega}} \cdot \hat{\mathbf{n}}\right)\, d\hat{\mathbf{\Omega}} \tag{2.18}$$

es ahora un escalar que da el número neto de neutrones que cruzan un área unitaria perpendicular a $\hat{\mathbf{n}}$ en la dirección positiva (figura 2.6).

Observación. La cantidad J_n es la resta de dos contribuciones

$$J_n(\mathbf{x}, E, t) = J_n^+(\mathbf{x}, E, t) - J_n^-(\mathbf{x}, E, t)$$

donde

$$\begin{aligned} J_n^+(\mathbf{x}, E, t) &= + \int_{\hat{\mathbf{\Omega}}\cdot\hat{\mathbf{n}}>0} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \left(\hat{\mathbf{\Omega}} \cdot \hat{\mathbf{n}}\right) d\hat{\mathbf{\Omega}} \\ J_n^-(\mathbf{x}, E, t) &= - \int_{\hat{\mathbf{\Omega}}\cdot\hat{\mathbf{n}}<0} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \left(\hat{\mathbf{\Omega}} \cdot \hat{\mathbf{n}}\right) d\hat{\mathbf{\Omega}} \end{aligned} \tag{2.19}$$

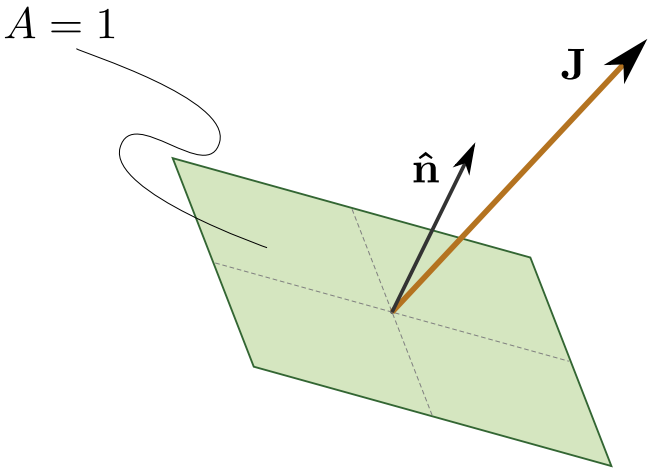


Figura 2.6.: Interpretación del producto del vector corriente $\mathbf{J}$ con el vector normal $\hat{\mathbf{n}}$ a una superficie como el número neto de neutrones que cruzan un área unitaria.

2.3. Transporte de neutrones

Habiendo introducido los conceptos básicos de "contabilidad" de neutrones, pasamos ahora a deducir las ecuaciones que gobiernan sus ritmos de

- aparición,
- desaparición, y
- transporte.

2.3.1. Operador de transporte

Consideremos un volumen finito $U \in \mathbb{R}^3$ arbitrario fijo en el espacio y consideremos ahora otro volumen $U'(t) \in \mathbb{R}^3$ que se mueve en una dirección fija $\hat{\mathbf{\Omega}}$ con una velocidad constante $v(E)$ correspondiente a una energía $E = 1/2 \cdot mv^2$ de un neutrón de masa m, de tal manera que en el instante t ambos volúmenes coincidan. En ese momento, la cantidad de neutrones con dirección $\hat{\mathbf{\Omega}}$ en torno al cono definido por $d\hat{\mathbf{\Omega}}$ y con energías entre E y $E + dE$ en el volumen $U \equiv U'(t)$ es

$$N_U(\hat{\mathbf{\Omega}}, E, t)\, d\hat{\mathbf{\Omega}}\, dE = \left[\int_{U \equiv U'(t)} N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d^3\mathbf{x}\right] d\hat{\mathbf{\Omega}}\, dE \tag{2.20}$$

Dado que la posición del dominio de integración cambia con el tiempo, la derivada total de esta magnitud con respecto al tiempo es la suma de una derivada parcial y una derivada material:

$$\frac{dN_U}{dt} = \frac{\partial N_U}{\partial t} + \lim_{\Delta t \to 0} \frac{1}{\Delta t}\left[\int_{U'(t+\Delta t)} N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d^3\mathbf{x} - \int_{U'(t)} N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d^3\mathbf{x}\right] \tag{2.21}$$

Notemos que el dominio U se mueve según

$$\lim_{\Delta t \to 0} U'(t + \Delta t) = \lim_{\Delta t \to 0} U'(t) + v(E) \cdot \hat{\mathbf{\Omega}} \cdot \Delta t$$

para cada punto $\mathbf{x} \in U'(t)$. Además, como ni $v(E)$ ni $\hat{\Omega}_i$ dependen de $\mathbf{x}$ ya que la velocidad es constante y la dirección está fija, entonces el cambio de coordenadas

$$\mathbf{x}' = \mathbf{x} + v(E) \cdot \hat{\mathbf{\Omega}} \cdot \Delta t$$

es unitario ya que

$$\begin{aligned}
\frac{\partial}{\partial x}\left[x + v(E)\,\hat{\Omega}_x \cdot \Delta t\right] &= 1 \\
\frac{\partial}{\partial y}\left[y + v(E)\,\hat{\Omega}_y \cdot \Delta t\right] &= 1 \\
\frac{\partial}{\partial z}\left[z + v(E)\,\hat{\Omega}_z \cdot \Delta t\right] &= 1
\end{aligned}$$

y entonces podemos hacer que el dominio de integración de ambas integrales de la ecuación 2.21 coincidan escribiendo

$$\begin{aligned}
\frac{dN_U}{dt} &= \frac{\partial N_U}{\partial t} + \lim_{\Delta t \to 0} \frac{1}{\Delta t}\left[\int_{U'(t)} N(\mathbf{x}', \hat{\mathbf{\Omega}}, E, t)\, d^3\mathbf{x} - \int_{U'(t)} N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d^3\mathbf{x}\right] \\
&= \frac{\partial N_U}{\partial t} + \lim_{\Delta t \to 0} \frac{1}{\Delta t}\left[\int_{U'(t)} N(\mathbf{x} + v(E) \cdot \hat{\mathbf{\Omega}} \cdot \Delta t, \hat{\mathbf{\Omega}}, E, t) - N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d^3\mathbf{x}\right]
\end{aligned}$$

El segundo término del miembro derecho es igual a $v(E)$ veces la derivada espacial direccional de la función $N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)$ en la dirección $\hat{\mathbf{\Omega}}$. En efecto, para $\hat{\mathbf{\Omega}}$, E y t fijos,

$$\begin{aligned}
&\lim_{\Delta t \to 0} N(x + v \cdot \Omega_x \cdot \Delta t, y + v \cdot \Omega_y \cdot \Delta t, z + v \cdot \Omega_y \cdot \Delta t) = \\
&\lim_{\Delta t \to 0} N(x, y, z) + \frac{\partial N}{\partial x} \cdot v \cdot \Omega_x \cdot \Delta t + \frac{\partial N}{\partial y} \cdot v \cdot \Omega_y \cdot \Delta t + \frac{\partial N}{\partial z} \cdot v \cdot \Omega_z \cdot \Delta t
\end{aligned}$$

Reordenando términos y dividiendo por Δt

$$\lim_{\Delta t \to 0} \frac{1}{\Delta t} \left[N(\mathbf{x} + v \cdot \hat{\mathbf{\Omega}} \cdot \Delta t) - N(\mathbf{x}) \right] = \text{grad}\left[N(\mathbf{x}) \right] \cdot v \cdot \hat{\mathbf{\Omega}}$$

Como habíamos supuesto que el dominio móvil $U'(t)$ coincide con el dominio fijo U en el instante t, es decir $U'(t) \equiv U$, entonces podemos escribir la derivada total de la cantidad de neutrones en U con respecto al tiempo como

$$\frac{dN_U}{dt} = \frac{\partial N_U}{\partial t} + \int_U \hat{\mathbf{\Omega}} \cdot v(E) \cdot \text{grad}\left[N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \right] d^3\mathbf{x}$$

Teniendo en cuenta la ecuación 2.20,

$$N_U(\hat{\mathbf{\Omega}}, E, t) = \int_{U \equiv U'(t)} N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d^3\mathbf{x}$$

y la definición 2.10 de flujo angular ψ

$$\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = v(E) \cdot N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)$$

tenemos

$$\frac{d}{dt} \int_U N(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d^3\mathbf{x} = \int_U \left\{ \frac{1}{v(E)} \frac{\partial \psi}{\partial t} + \hat{\mathbf{\Omega}} \cdot \text{grad}\left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \right] \right\} d^3\mathbf{x}$$

Como el dominio U es arbitrario, entonces

$$\frac{dN}{dt} = \frac{1}{v(E)} \frac{\partial \psi}{\partial t} + \hat{\mathbf{\Omega}} \cdot \text{grad}\left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \right] \tag{2.22}$$

Observación. El operador gradiente se aplica sólo sobre las componentes espaciales, es decir

$$\text{grad}\left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \right] = \begin{bmatrix} \dfrac{\partial \psi}{\partial x} \\ \dfrac{\partial \psi}{\partial y} \\ \dfrac{\partial \psi}{\partial z} \end{bmatrix} \tag{2.23}$$

2.3.2. Operador de desapariciones

Cuando un neutrón reacciona con un núcleo blanco, el neutrón incidente puede...

a. ser dispersado, o
b. generar una fisión, o
c. ser capturado por el núcleo blanco.

En los últimos dos casos, el neutrón incidente realmente "desaparece" físicamente. En el caso b, vuelven a nacer $\nu(E)$ neutrones nuevos: una fracción $(1-\beta)$ instantáneamente y β en forma retardada durante la cadena de decaimiento de los productos de fisión. En el primer caso, el neutrón incidente nunca desaparece sino que cambia su energía y su dirección de vuelo. De todas maneras, conceptualmente es posible pensar que aún en un proceso de scattering el neutrón incidente de energía E y dirección $\hat{\mathbf{\Omega}}$ desaparece y se emite exactamente un neutrón nuevo con una energía E' y una dirección $\hat{\mathbf{\Omega}}'$ según la ley de scattering de la sección eficaz diferencial $\Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E')$ dada por la definición 2.2.

Con esta idea, cualquier reacción nuclear genera una desaparición del neutrón incidente al mismo tiempo que los casos a y b producen nuevos neutrones. Recordando el corolario 2.5, la tasa o el ritmo de reacciones de cualquier tipo por unidad de volumen y unidad de tiempo es el producto de la sección eficaz total $\Sigma_t(\mathbf{x}, E)$ por el flujo escalar $\phi(\mathbf{x}, E, t)$. Entonces la tasa de desaparición de neutrones por unidad de volumen y de tiempo es

$$\Sigma_t(\mathbf{x}, E) \cdot \phi(\mathbf{x}, E, t)\, d^3\mathbf{x} = \Sigma_t(\mathbf{x}, E) \cdot \int_{4\pi} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d\hat{\mathbf{\Omega}}\, d^3\mathbf{x} \tag{2.24}$$

2.3.3. Operador de producciones

Pasamos ahora a estudiar las posibles maneras en las que pueden aparecer neutrones. Éstos pueden aparecer debido a uno de los siguiente tres mecanismos, que analizamos en las secciones que siguen:

1. scattering,
2. fisión, o
3. fuentes externas.

2.3.3.1. Fuentes por scattering

A la luz de la discusión de la sección 2.3.2 podemos decir que un neutrón que viajando con una energía E y una dirección $\hat{\mathbf{\Omega}}$ sufre una colisión de scattering en el punto $\mathbf{x}$ es absorbido por el núcleo blanco. Al mismo tiempo, emerge como un nuevo neutrón con una energía E' y una dirección $\hat{\mathbf{\Omega}}'$. Esta fuente q_s debe ser entonces igual al producto de la probabilidad por unidad de longitud de recorrido de neutrones que viajando en con una energía E en una dirección $\hat{\mathbf{\Omega}}$ colisionen con un núcleo blanco en el punto $\mathbf{x}$ y como resultado adquieren una dirección de viaje $\hat{\mathbf{\Omega}}'$ y una energía E' (recordar los conceptos introducidos en la sección 2.1.1) por la cantidad de longitudes lineales viajadas,

teniendo en cuenta todos los posibles valores de $\hat{\mathbf{\Omega}}$ y de E. Es decir, teniendo en cuenta la definición 2.2 y el corolario 2.3, la fuente de neutrones debida scattering que da como resultado neutrones de energías E' y direcciones $\hat{\mathbf{\Omega}}'$ es

$$q_s(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t) = \int_0^\infty \int_{4\pi} \Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E') \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d\hat{\mathbf{\Omega}}\, dE$$

Dado que en general estamos interesados en el ritmo en los que los neutrones *nacen* con energía E y dirección $\hat{\mathbf{\Omega}}$ en el dominio, podemos invertir las variables primadas para obtener la expresión de la fuente de neutrones debidas a scattering por unidad de volumen y de tiempo

$$q_s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \int_0^\infty \int_{4\pi} \Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}}' \to \hat{\mathbf{\Omega}}, E' \to E) \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t)\, d\hat{\mathbf{\Omega}}'\, dE' \tag{2.25}$$

2.3.3.2. Fuentes por fisiones

Ya hemos discutido brevemente en la sección 2.1.3 (y lo haremos más en detalle en la sección 3.5), debemos calcular la fuente de fisión ligeramente diferente si estamos resolviendo problemas

a. transitorios
b. estacionarios
    i. con fuentes independientes
    ii. sólo con fuentes de fisión

Sin pérdida de generalidad, para fijar ideas supongamos que desde el punto de vista de la fisión el problema es estacionario tanto con fuentes por fisión como con fuentes independientes. De manera análoga a la fuente por scattering, la tasa de generación de neutrones debidas a fisión debido a un neutrón incidente con una dirección $\hat{\mathbf{\Omega}}$ y una energía E es el producto del número probable de nacimientos $\nu(E)$ multiplicada por la probabilidad de que dicho neutrón incidente genere una fisión en el punto $\mathbf{x}$ por unidad de longitud de recorrido multiplicada por la cantidad de longitudes lineales viajadas, teniendo en cuenta todos los posibles valores de direcciones $\hat{\mathbf{\Omega}}$ y energías E incidentes:

$$q_f(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t) = \nu(E) \cdot \int_0^\infty \int_{4\pi} \Sigma_f(\mathbf{x}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E \to E') \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d\hat{\mathbf{\Omega}}\, dE$$

Recordando la ecuación ecuación 2.14, invirtiendo las variables primadas y escribiendo el producto $\nu(E) \cdot \Sigma_{f_t}(\mathbf{x}, E)$ como una única función $\nu\Sigma_f(\mathbf{x}.E)$, tenemos

$$q_f(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \frac{\chi(E)}{4\pi} \int_0^\infty \int_{4\pi} \nu\Sigma_f(\mathbf{x}, E') \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t)\, d\hat{\mathbf{\Omega}}'\, dE' \tag{2.26}$$

2.3.3.3. Fuentes independientes

Por último, para no perder generalidad tenemos que tener en cuenta las fuentes externas de neutrones, es decir aquellas que no provienen de la fisión de materiales presentes en el núcleo sino de otras fuentes totalmente independientes como por ejemplo una fuente de americio-berilio. Matemáticamente, las modelamos como la integral sobre el volumen U de una función arbitraria

$$s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \tag{2.27}$$

del espacio, la dirección, la energía y el tiempo que representa la cantidad de neutrones emitidos con energía E en el punto $\mathbf{x}$ con dirección $\hat{\mathbf{\Omega}}$ en el instante t.

2.3.4. La ecuación de transporte

La conservación de neutrones implica que la derivada temporal total de la población de neutrones en la posición $\mathbf{x}$ viajando en la dirección $\hat{\mathbf{\Omega}}$ con una energía E debe ser igual a la diferencia entre la suma de las tres fuentes de las ecuación 2.25 y la tasa de desapariciones ecuación 2.24, tenemos

$$\begin{gathered} \frac{1}{v}\frac{\partial \psi}{\partial t} + \hat{\mathbf{\Omega}} \cdot \operatorname{grad}\left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\right] = \\ q_s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) + q_f(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) + s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) - \Sigma_t(\mathbf{x}, E) \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \end{gathered} \tag{2.28}$$

Escribiendo explícitamente los dos términos de fuente por scattering y fisión, pasando el término negativo de desapariciones al miembro izquierdo y teniendo en cuenta que la relación entre velocidad y energía es la clásica $E = 1/2 \cdot mv^2$, llegamos a la famosa *ecuación de transporte de neutrones*

$$\begin{aligned} &\sqrt{\frac{m}{2E}}\frac{\partial}{\partial t}\left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\right] + \hat{\mathbf{\Omega}} \cdot \operatorname{grad}\left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\right] + \Sigma_t(\mathbf{x}, E) \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \\ &\qquad \int_0^\infty \int_{4\pi} \Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}}' \to \hat{\mathbf{\Omega}}, E' \to E) \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t)\, d\hat{\mathbf{\Omega}}'\, dE' \\ &+ \frac{\chi(E)}{4\pi} \int_0^\infty \int_{4\pi} \nu\Sigma_f(\mathbf{x}, E') \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t)\, d\hat{\mathbf{\Omega}}'\, dE' + s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \end{aligned} \tag{2.29}$$

que es una ecuación integro-diferencial hiperbólica en derivadas parciales de primer orden tanto sobre el espacio (notar que el operador gradiente opera sólo sobre las coordenadas espaciales $\mathbf{x}$ según la ecuación 2.23) como sobre el tiempo para la incógnita ψ sobre un espacio de fases multidimensional que incluye

1. un dominio espacial $U \in \mathbb{R}^3 / \mathbf{x} \in U$,
2. el versor dirección $\hat{\mathbf{\Omega}} = [\Omega_x, \Omega_y, \Omega_z] / \Omega_x^2 + \Omega_y^2 + \Omega_z^2 = 1$,
3. la energía $E \in (0, \infty)$, y
4. el tiempo $t \in [0, \infty)$.

Los datos necesarios para resolver la ecuación de transporte de neutrones son:

- Las secciones eficaces Σ_t y $\nu\Sigma_f$ como función del espacio $\mathbf{x}$ y de la energía E' del neutrón incidente.
- El espectro de fisión χ en función de la energía E del neutrón emitido, en caso de que $\nu\Sigma_f \neq 0$.
- La sección eficaz diferencial de scattering Σ_s como función tanto de la energía E' del neutrón incidente como de la energía E del neutrón saliente, y del ángulo de scattering entre la dirección entrante $\hat{\mathbf{\Omega}}'$ y la dirección saliente $\hat{\mathbf{\Omega}}$.
  - Esta dependencia es usualmente dada como coeficientes Σ_{s_ℓ} de la expansión en polinomios de Legendre para $\ell = 0, \dots, L$ sobre el escalar $\mu = \hat{\mathbf{\Omega}}' \cdot \hat{\mathbf{\Omega}}$.
  - Para scattering isótropo en el marco de referencia del reactor, el único coeficiente diferente de cero es Σ_{s_0} correspondiente a $\ell = 0$.
- La fuente independiente de neutrones s como función del espacio $\mathbf{x}$, la energía E y de la dirección $\hat{\mathbf{\Omega}}$.
  - Si no hay fuentes externas, este término es cero.
- El parámetro constante m, que es la masa en reposo del neutrón
  - Este parámetro es sólo necesario en cálculos transitorios. Los cálculos estacionarios son independientes de m.

2.3.5. Armónicos esféricos y polinomios de Legendre

Prestemos atención al término de fuente por scattering dado por la ecuación 2.25

$$q_s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \int_0^\infty \int_{4\pi} \Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}}' \to \hat{\mathbf{\Omega}}, E' \to E) \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t)\, d\hat{\mathbf{\Omega}}'\, dE' \tag{2.25}$$

y supongamos que conocemos los coeficientes Σ_{s_ℓ} de la expansión en polinomios de Legendre de la sección eficaz diferencial de scattering sobre el coseno $\mu = \hat{\mathbf{\Omega}} \cdot \hat{\mathbf{\Omega}}'$ del ángulo de dispersión según la ecuación 2.3

$$\Sigma_s(\mathbf{x}, \mu, E \to E') = \sum_{\ell=0}^{\infty} \frac{2\ell + 1}{2} \Sigma_{s_\ell}(\mathbf{x}, E \to E') \cdot P_\ell(\mu) \tag{2.3}$$

Para poder usar estos coeficientes en la fuente de la ecuación 2.25 debemos recordar el resultado de la ecuación 2.9, ya que para cada μ hay 2π posibles ángulos $\hat{\mathbf{\Omega}}'$:

$$\begin{aligned}\Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}}' \to \hat{\mathbf{\Omega}}, E' \to E) &= \frac{1}{2\pi} \cdot \Sigma_s(\mathbf{x}, \mu, E' \to E) \\ &= \frac{1}{2\pi} \cdot \sum_{\ell=0}^{\infty} \frac{2\ell + 1}{2} \cdot \Sigma_{s_\ell}(\mathbf{x}, E' \to E) \cdot P_\ell(\mu)\end{aligned}$$

Ahora sí podemos reemplazar esta expansión de $\Sigma_s(\mathbf{x}, \hat{\mathbf{\Omega}}' \to \hat{\mathbf{\Omega}}, E' \to E)$ en el término de fuentes de la ecuación 2.25,

$$
\begin{aligned}
q_s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) &= \int_0^\infty \left[\int_{4\pi} \frac{1}{2\pi} \sum_{\ell=0}^{\infty} \frac{2\ell+1}{2} \cdot \Sigma_{s_\ell}(\mathbf{x}, E' \to E) \cdot P_\ell(\mu) \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t)\, d\hat{\mathbf{\Omega}}' \right] dE' \\
&= \int_0^\infty \sum_{\ell=0}^{\infty} \frac{2\ell+1}{4\pi} \cdot \left[\Sigma_{s_\ell}(\mathbf{x}, E' \to E) \int_{4\pi} P_\ell(\hat{\mathbf{\Omega}} \cdot \hat{\mathbf{\Omega}}') \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t)\, d\hat{\mathbf{\Omega}}' \right] dE'
\end{aligned}
\tag{2.30}
$$

Si bien esta expresión ya es suficiente para evaluar el término de scattering cuando tenemos su desarrollo de Legendre, podemos

1. ahondar un poco más en la estructura de la ecuación de transporte, y
2. obtener una expresión computacionalmente más eficiente

expandiendo en una base apropiada el flujo angular ψ, de la misma manera en la que desarrollamos Σ_s en una serie de polinomios de Legendre sobre el parámetro μ.

Para ello, notamos que ψ depende angularmente de un versor dirección $\hat{\mathbf{\Omega}} = [\hat{\Omega}_x\, \hat{\Omega}_y\, \hat{\Omega}_z]^T$ (u $\hat{\mathbf{\Omega}}'$ en el caso de la ecuación 2.30). Esta vez, la base de expansión apropiada no son los polinomios de Legendre (que toman un único argumento escalar μ) sino la generada[11] por los armónicos esféricos reales, ilustrados en la figura 2.7.

Teorema 2.3 (expansión en armónicos esféricos reales). *Cualquier función $f(\hat{\Omega}_x, \hat{\Omega}_y, \hat{\Omega}_z)$ de cuadrado integrable con*

$$\hat{\Omega}_x^2 + \hat{\Omega}_y^2 + \hat{\Omega}_z^2 = 1$$

puede ser escrita como la suma doble de un coeficiente f_ℓ^m por el armónico esférico normalizado real $Y_\ell^m\left(\hat{\mathbf{\Omega}}\right)$ de grado $\ell \geq 0$ y orden m:

$$f(\hat{\Omega}_x, \hat{\Omega}_y, \hat{\Omega}_z) = \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{\ell} f_\ell^m \cdot Y_\ell^m(\hat{\Omega}_x, \hat{\Omega}_y, \hat{\Omega}_z)$$

donde para cada grado ℓ el orden m es tal que

$$-\ell \leq m \leq \ell$$

Definición 2.13. Los primeros armónicos esféricos reales (figura 2.7) son

[11] Del inglés *spanned*.

$$Y_0^0(\hat{\Omega}_x, \hat{\Omega}_y, \hat{\Omega}_z) = \sqrt{\frac{1}{4\pi}}$$

$$\begin{aligned}
Y_1^{-1}(\hat{\Omega}_x, \hat{\Omega}_y, \hat{\Omega}_z) &= \sqrt{\frac{3}{4\pi}} \cdot \hat{\Omega}_y \\
Y_1^{0}(\hat{\Omega}_x, \hat{\Omega}_y, \hat{\Omega}_z) &= \sqrt{\frac{3}{4\pi}} \cdot \hat{\Omega}_z \\
Y_1^{+1}(\hat{\Omega}_x, \hat{\Omega}_y, \hat{\Omega}_z) &= \sqrt{\frac{3}{4\pi}} \cdot \hat{\Omega}_x
\end{aligned}$$

$$\begin{aligned}
Y_2^{-2}(\hat{\Omega}_x, \hat{\Omega}_y, \hat{\Omega}_z) &= \sqrt{\frac{15}{4\pi}} \cdot \hat{\Omega}_x \cdot \hat{\Omega}_y \\
Y_2^{-1}(\hat{\Omega}_x, \hat{\Omega}_y, \hat{\Omega}_z) &= \sqrt{\frac{15}{4\pi}} \cdot \hat{\Omega}_y \cdot \hat{\Omega}_z \\
Y_2^{0}(\hat{\Omega}_x, \hat{\Omega}_y, \hat{\Omega}_z) &= \sqrt{\frac{5}{16\pi}} \cdot \left(-\hat{\Omega}_x^2 - \hat{\Omega}_y^2 + 2 \cdot \hat{\Omega}_z^2\right) \\
Y_2^{+1}(\hat{\Omega}_x, \hat{\Omega}_y, \hat{\Omega}_z) &= \sqrt{\frac{15}{4\pi}} \cdot \hat{\Omega}_z \cdot \hat{\Omega}_x \\
Y_2^{+2}(\hat{\Omega}_x, \hat{\Omega}_y, \hat{\Omega}_z) &= \sqrt{\frac{15}{16\pi}} \cdot \left(\hat{\Omega}_x^2 - \hat{\Omega}_y^2\right)
\end{aligned}$$

Teorema 2.4 (ortonormalidad de los armónicos esféricos reales). *Los armónicos esféricos reales son ortonormales, es decir*

$$\int_{4\pi} Y_\ell^m(\hat{\mathbf{\Omega}}) \cdot Y_{\ell'}^{m'}(\hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}} = \delta_{\ell\ell'} \cdot \delta_{mm'} \begin{cases} 1 & \textit{si } \ell = \ell' \wedge m = m' \\ 0 & \textit{si } \ell \neq \ell' \vee m \neq m' \end{cases}$$

Corolario 2.6. *Los coeficientes f_ℓ^m son iguales a*

$$f_\ell^m(\mathbf{x}, E, t) = \int_{4\pi} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \cdot Y_\ell^m(\hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}}$$

Teorema 2.5 (de adición). *Los armónicos esféricos se relacionan con los polinomios de Legendre como*

$$P_\ell(\hat{\mathbf{\Omega}} \cdot \hat{\mathbf{\Omega}}') = \frac{4\pi}{2\ell + 1} \sum_{m=-\ell}^{+\ell} Y_\ell^m(\hat{\mathbf{\Omega}}) \cdot Y_\ell^m(\hat{\mathbf{\Omega}}')$$

Observación. Para $\ell = 0$ tenemos $P_0(\mu) = 1$ e $Y_0^0(\hat{\mathbf{\Omega}}) = \sqrt{1/4\pi}$. En efecto,

$$P_0(\mu) = 1 = \frac{4\pi}{2 \cdot 0 + 1} \cdot \sqrt{\frac{1}{4\pi}} \cdot \sqrt{\frac{1}{4\pi}} = 1$$

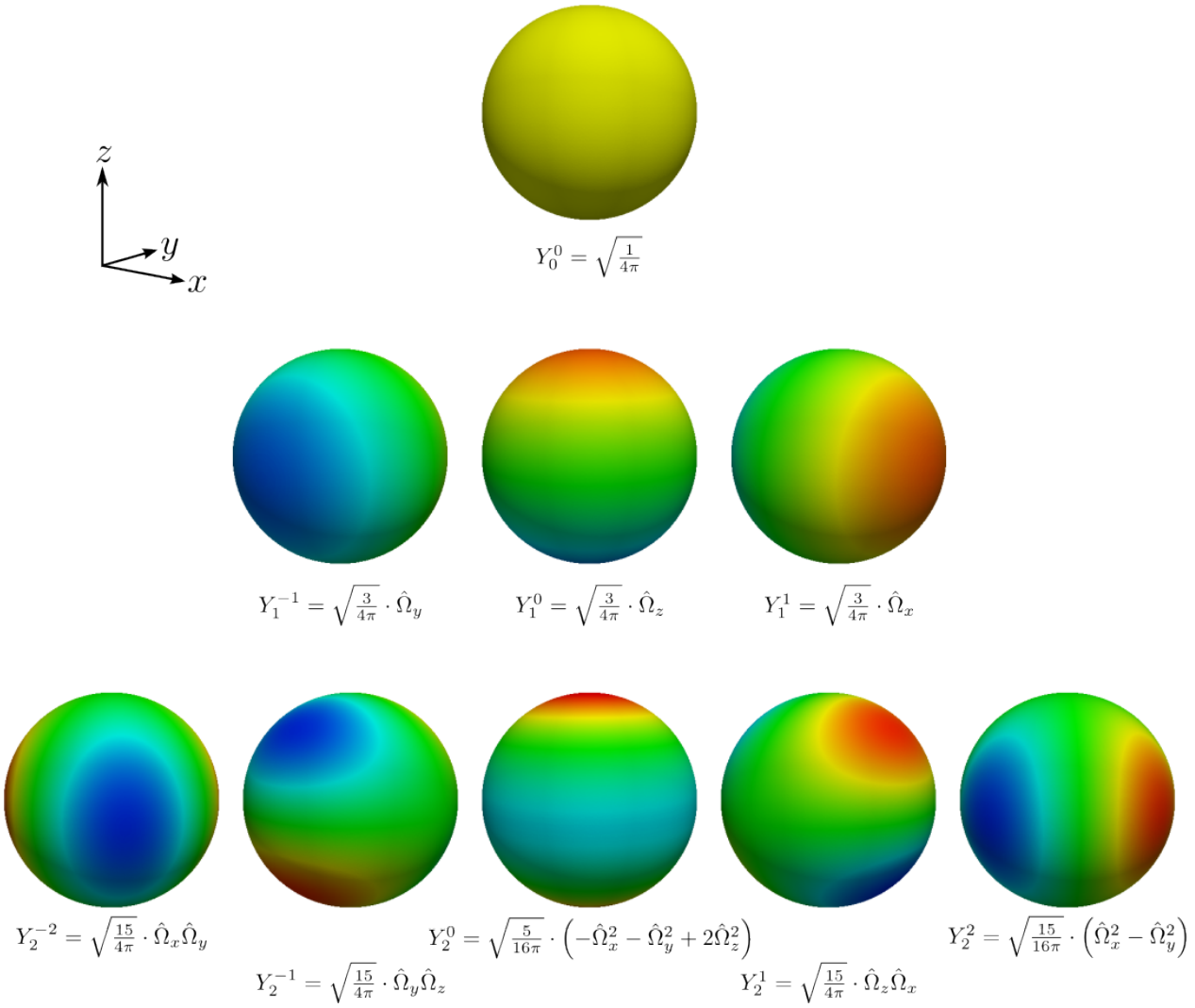


Figura 2.7.: Representación gráfica de los primeros nueve armónicos esféricos reales (definición 2.13). En la sección 4.3.1 explicamos cómo generamos esta figura.

Si en el teorema 2.3 hacemos f igual al flujo angular ψ entonces podemos escribirlo como una suma doble sobre ℓ y sobre m del producto de un coeficiente que depende del espacio $\mathbf{x}$, de la energía E y del tiempo t (pero no de la dirección $\hat{\mathbf{\Omega}}$) por el armónico esférico de grado ℓ y orden m, que no depende ni del espacio $\mathbf{x}$ ni de la energía E ni del tiempo t (pero sí de la dirección $\hat{\mathbf{\Omega}}$):

$$\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{+\ell} \Psi_\ell^m(\mathbf{x}, E, t) \cdot Y_\ell^m\left(\hat{\mathbf{\Omega}}\right) \tag{2.31}$$

Volvamos entonces al término de scattering q_s dado por la ecuación 2.30 y reemplacemos $P_\ell(\hat{\mathbf{\Omega}} \cdot \hat{\mathbf{\Omega}}')$ en la integral sobre $\hat{\mathbf{\Omega}}'$ por el teorema 2.5:

$$\begin{aligned} &q_s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \\ &\int_0^\infty \sum_{\ell=0}^{\infty} \frac{2\ell+1}{4\pi} \left[\Sigma_{s_\ell}(\mathbf{x}, E' \to E) \int_{4\pi} \frac{4\pi}{2\ell+1} \sum_{m=-\ell}^{+\ell} Y_\ell^m(\hat{\mathbf{\Omega}}) \cdot Y_\ell^m(\hat{\mathbf{\Omega}}') \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t)\, d\hat{\mathbf{\Omega}}' \right] dE' \\ &= \int_0^\infty \sum_{\ell=0}^{\infty} \left[\Sigma_{s_\ell}(\mathbf{x}, E' \to E) \sum_{m=-\ell}^{+\ell} Y_\ell^m(\hat{\mathbf{\Omega}}) \int_{4\pi} Y_\ell^m(\hat{\mathbf{\Omega}}') \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t)\, d\hat{\mathbf{\Omega}}' \right] dE' \end{aligned}$$

La integral sobre $d\hat{\mathbf{\Omega}}'$ dentro del corchete es justamente el coeficiente Ψ_ℓ^m de la expansión en armónicos esféricos del flujo angular ψ dado por el corolario 2.6. Luego

$$q_s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \int_0^\infty \sum_{\ell=0}^{\infty} \left[\Sigma_{s_\ell}(\mathbf{x}, E' \to E) \sum_{m=-\ell}^{+\ell} \Psi_\ell^m(\mathbf{x}, E', t) \cdot Y_\ell^m(\hat{\mathbf{\Omega}}) \right] dE' \tag{2.32}$$

Esta ecuación 2.32 refleja la forma en la que incide la fuente de scattering en el balance global de neutrones: el modo ℓ de la expansión en polinomios de Legendre de la sección diferencial Σ_s de scattering contribuye sólo a través de los modos de grado ℓ de la expansión en armónicos esféricos del flujo angular ψ. En particular, para scattering isótropo sólo el término para $\ell = 0$ y $m = 0$ contribuye a la fuente de scattering q_s. De la misma manera, para scattering linealmente anisótropo además sólo contribuyen los tres términos con $\ell = 1$ y $m = -1, 0, +1$.

Prestemos atención ahora al coeficiente Ψ_0^0. Ya que por un lado $Y_0^0 = \sqrt{1/4\pi}$ (definición 2.13) mientras que por otro la integral del flujo angular ψ con respecto a $\hat{\mathbf{\Omega}}$ sobre todas las direcciones es igual al flujo escalar (definición 2.11), entonces

$$\Psi_0^0(\mathbf{x}, E, t) = \int_{4\pi} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \cdot Y_0^0(\hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}} = \sqrt{\frac{1}{4\pi}} \cdot \phi(\mathbf{x}, E, t)$$

De esta manera, podemos escribir la suma de la ecuación 2.31 con el primer término de la expansión del flujo angular ψ escrito explícitamente como

$$\begin{aligned}\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t) &= \sum_{\ell=0}^{\infty}\sum_{m=-\ell}^{+\ell}\Psi_\ell^m(\mathbf{x},E,t)\cdot Y_\ell^m(\hat{\mathbf{\Omega}}) \\ &= \sqrt{\frac{1}{4\pi}}\cdot\phi(\mathbf{x},E,t)\cdot\sqrt{\frac{1}{4\pi}} + \sum_{\ell=1}^{\infty}\sum_{m=-\ell}^{+\ell}\Psi_\ell^m(\mathbf{x},E,t)\cdot Y_\ell^m(\hat{\mathbf{\Omega}}) \\ &= \frac{\phi(\mathbf{x},E,t)}{4\pi} + \sum_{\ell=1}^{\infty}\sum_{m=-\ell}^{+\ell}\Psi_\ell^m(\mathbf{x},E,t)\cdot Y_\ell^m(\hat{\mathbf{\Omega}})\end{aligned}$$

Más aún, según la definición 2.13, los coeficientes de grado $\ell = 1$ son

$$\begin{aligned}\Psi_1^{-1}(\mathbf{x},E,t) &= \int_{4\pi}\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\cdot\sqrt{\tfrac{3}{4\pi}}\cdot\hat{\Omega}_y\,d\hat{\mathbf{\Omega}} = \sqrt{\tfrac{3}{4\pi}}\cdot J_y(\mathbf{x},E,t) \\ \Psi_1^{0}(\mathbf{x},E,t) &= \int_{4\pi}\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\cdot\sqrt{\tfrac{3}{4\pi}}\cdot\hat{\Omega}_z\,d\hat{\mathbf{\Omega}} = \sqrt{\tfrac{3}{4\pi}}\cdot J_z(\mathbf{x},E,t) \\ \Psi_1^{1+}(\mathbf{x},E,t) &= \int_{4\pi}\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\cdot\sqrt{\tfrac{3}{4\pi}}\cdot\hat{\Omega}_x\,d\hat{\mathbf{\Omega}} = \sqrt{\tfrac{3}{4\pi}}\cdot J_x(\mathbf{x},E,t)\end{aligned}$$

donde en la última igualdad hemos empleado la definición 2.12 del vector corriente

$$\mathbf{J}(\mathbf{x},E,t) = \int_{4\pi}\left[\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\cdot\hat{\mathbf{\Omega}}\right]\,d\hat{\mathbf{\Omega}} = \begin{bmatrix}J_x\\J_y\\J_z\end{bmatrix}$$

y la ecuación 2.18.

Observación. El vector corriente $\mathbf{J}(\mathbf{x},E,t)$ depende del espacio $\mathbf{x}$, la energía E y el tiempo t pero *no* del ángulo $\hat{\mathbf{\Omega}}$.

Podemos escribir también los tres términos correspondiente a $\ell = 1$ de la expansión del flujo angular ψ explícitamente como tres veces (sobre 4π) el producto interno entre el vector corriente $\mathbf{J}(\mathbf{x},E,t)$ y la dirección $\hat{\mathbf{\Omega}}$ de forma tal que

$$\begin{aligned}\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t) &= \frac{\phi(\mathbf{x},E,t)}{4\pi} + \sum_{\ell=1}^{\infty}\sum_{m=-\ell}^{+\ell}\Psi_\ell^m(\mathbf{x},E,t)\cdot Y_\ell^m(\hat{\mathbf{\Omega}}) \\ &= \frac{1}{4\pi}\left[\phi(\mathbf{x},E,t) + 3\cdot\left(\mathbf{J}(\mathbf{x},E,t)\cdot\hat{\mathbf{\Omega}}\right)\right] + \sum_{\ell=2}^{\infty}\sum_{m=-\ell}^{+\ell}\Psi_\ell^m(\mathbf{x},E,t)\cdot Y_\ell^m(\hat{\mathbf{\Omega}})\end{aligned} \tag{2.33}$$

Como comprobación, verificamos que a partir de esta expresión podemos recuperar el flujo escalar integrando con respecto a $\hat{\mathbf{\Omega}}$ sobre 4π

$$\begin{aligned}\phi(\mathbf{x},E,t) &= \int_{4\pi} \psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\,d\hat{\mathbf{\Omega}} \\ &= \frac{1}{4\pi}\int_{4\pi}\left[\phi(\mathbf{x},E,t) + \left(3\cdot\mathbf{J}(\mathbf{x},E,t)\cdot\hat{\mathbf{\Omega}}\right) + \sum_{\ell=2}^{\infty}\sum_{m=-\ell}^{+\ell}\Psi_\ell^m(\mathbf{x},E,t)\cdot Y_\ell^m(\hat{\mathbf{\Omega}})\right] d\hat{\mathbf{\Omega}}\end{aligned} \tag{2.34}$$

Por un lado, la integral del segundo término $3(\mathbf{J}\cdot\hat{\mathbf{\Omega}})$ sobre $d\hat{\mathbf{\Omega}}$ es cero. En efecto,

$$\int_{4\pi} 3\cdot\mathbf{J}(\mathbf{x},E,t)\cdot\hat{\mathbf{\Omega}}\,d\hat{\mathbf{\Omega}} = 3\cdot\mathbf{J}(\mathbf{x},E,t)\cdot\int_{4\pi}\hat{\mathbf{\Omega}}\,d\hat{\mathbf{\Omega}} = 3\cdot\mathbf{J}(\mathbf{x},E,t)\int_{4\pi}\begin{bmatrix}\Omega_x\\ \Omega_y\\ \Omega_z\end{bmatrix} d\hat{\mathbf{\Omega}} = 0$$

Para analizar los siguientes términos para $\ell \geq 2$ necesitamos tener en cuenta el siguiente teorema.

Teorema 2.6. *La integral sobre la esfera unitaria del armónico esférico Y_ℓ^m de grado ℓ y orden m es*

$$\int_{4\pi} Y_\ell^m(\hat{\mathbf{\Omega}})\,d\hat{\mathbf{\Omega}} = \begin{cases}\sqrt{4\pi} & \textit{para } \ell = 0 \wedge m = 0\\ 0 & \textit{de otra manera}\end{cases}$$

Demostración. Tomemos $\ell' = 0$ y $m' = 0$ en el teorema 2.4 y reemplacemos la definición 2.13 para $Y_0^0 = 1/\sqrt{4\pi}$

$$\begin{aligned}\int_{4\pi} Y_\ell^m(\hat{\mathbf{\Omega}})\cdot Y_0^0(\hat{\mathbf{\Omega}})\,d\hat{\mathbf{\Omega}} &= \delta_{\ell 0}\delta_{m0}\\ \int_{4\pi} Y_\ell^m(\hat{\mathbf{\Omega}})\cdot\sqrt{\frac{1}{4\pi}}\,d\hat{\mathbf{\Omega}} &= \delta_{\ell 0}\delta_{m0}\\ \int_{4\pi} Y_\ell^m(\hat{\mathbf{\Omega}})\,d\hat{\mathbf{\Omega}} &= \sqrt{4\pi}\cdot\delta_{\ell 0}\cdot\delta_{m0}\end{aligned}$$

□

En virtud del teorema 2.6, los términos de la sumatoria sobre $\ell \geq 2$ en la ecuación 2.34 tampoco contribuyen a la integral por lo que

$$\phi(\mathbf{x},E,t) = \int_{4\pi}\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\,d\hat{\mathbf{\Omega}} = \frac{1}{4\pi}\int_{4\pi}\phi(\mathbf{x},E,t)\,d\hat{\mathbf{\Omega}} = \phi(\mathbf{x},E,t)$$

Teorema 2.7. *La integral del producto de dos cosenos dirección individuales*

$$\int_{4\pi}\hat{\Omega}_i\cdot\hat{\Omega}_j\,d\hat{\mathbf{\Omega}} = \frac{4\pi}{3}\cdot\delta_{ij}$$

para $i = x,y,z$ y $j = x,y,z$.

Teorema 2.8. *La integral del producto de una cantidad impar de cosenos es cero*

$$\int_{4\pi} \hat{\Omega}_x^r \cdot \hat{\Omega}_y^s \cdot \hat{\Omega}_z^t \, d\hat{\mathbf{\Omega}} = 0 \quad \textit{si } r, s \textit{ o } t \textit{ es impar}$$

Podemos demostrar entonces que con la definición 2.12 recuperamos además el vector corriente.

Demostración. En efecto,

$$\begin{aligned}
\mathbf{J}(\mathbf{x}, E, t) &= \int_{4\pi} \left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \cdot \hat{\mathbf{\Omega}}\right] \, d\hat{\mathbf{\Omega}} \\
&= \frac{1}{4\pi} \int_{4\pi} \Big\{ \phi(\mathbf{x}, E, t) \cdot \hat{\mathbf{\Omega}} + 3 \cdot \left[\mathbf{J}(\mathbf{x}, E, t) \cdot \hat{\mathbf{\Omega}}\right] \cdot \hat{\mathbf{\Omega}} \\
&\qquad + \sum_{\ell=2}^{\infty} \sum_{m=-\ell}^{+\ell} \Psi_\ell^m(\mathbf{x}, E, t) \cdot \left[Y_\ell^m(\hat{\mathbf{\Omega}}) \cdot \hat{\mathbf{\Omega}}\right] \Big\} \, d\hat{\mathbf{\Omega}}
\end{aligned} \tag{2.35}$$

El primer término $\phi \cdot \hat{\mathbf{\Omega}}$ se anula porque por un lado ϕ no depende de $\hat{\mathbf{\Omega}}$ y por otro, como ya demostramos, $\int_{4\pi} \hat{\mathbf{\Omega}} \, d\hat{\mathbf{\Omega}} = 0$. Pero además los factores $Y_\ell^m \cdot \hat{\mathbf{\Omega}}$ dentro de la sumatoria doble también se anulan porque cualquier armónico Y_ℓ^m con $\ell > 1$ es ortogonal con respecto a los tres armónicos Y_1^m de orden $\ell = 1$, que a su vez son proporcionales a $\hat{\mathbf{\Omega}}$:

$$\begin{bmatrix} Y_1^{+1}(\hat{\mathbf{\Omega}}) \\ Y_1^{-1}(\hat{\mathbf{\Omega}}) \\ Y_1^{0}(\hat{\mathbf{\Omega}}) \end{bmatrix} = \sqrt{\frac{3}{4\pi}} \cdot \begin{bmatrix} \hat{\Omega}_x \\ \hat{\Omega}_y \\ \hat{\Omega}_z \end{bmatrix} = \sqrt{\frac{3}{4\pi}} \cdot \hat{\mathbf{\Omega}} \tag{2.36}$$

Entonces

$$\begin{aligned}
\mathbf{J}(\mathbf{x}, E, t) &= \frac{3}{4\pi} \int_{4\pi} \left(J_x \hat{\Omega}_x + J_y \hat{\Omega}_y + J_z \hat{\Omega}_z\right) \cdot \begin{bmatrix} \hat{\Omega}_x \\ \hat{\Omega}_y \\ \hat{\Omega}_z \end{bmatrix} \, d\hat{\mathbf{\Omega}} \\
&= \frac{3}{4\pi} \int_{4\pi} \begin{bmatrix} J_x \hat{\Omega}_x \hat{\Omega}_x + J_y \hat{\Omega}_y \hat{\Omega}_x + J_z \hat{\Omega}_z \hat{\Omega}_x \\ J_x \hat{\Omega}_x \hat{\Omega}_y + J_y \hat{\Omega}_y \hat{\Omega}_y + J_z \hat{\Omega}_z \hat{\Omega}_y \\ J_x \hat{\Omega}_x \hat{\Omega}_z + J_y \hat{\Omega}_y \hat{\Omega}_z + J_z \hat{\Omega}_z \hat{\Omega}_z \end{bmatrix} \, d\hat{\mathbf{\Omega}}
\end{aligned}$$

Usando el teorema 2.7,

$$\begin{aligned}
\mathbf{J}(\mathbf{x}, E, t) &= \frac{3}{4\pi} \int_{4\pi} \begin{bmatrix} J_x \cdot \frac{4\pi}{3} + J_y \cdot 0 + J_z \cdot 0 \\ J_x \cdot 0 + J_y \cdot \frac{4\pi}{3} + J_z \cdot 0 \\ J_x \cdot 0 + J_y \cdot 0 + J_z \cdot \frac{4\pi}{3} \end{bmatrix} \, d\hat{\mathbf{\Omega}} \\
&= \frac{3}{4\pi} \int_{4\pi} \begin{bmatrix} \frac{4\pi}{3} J_x \\ \frac{4\pi}{3} J_y \\ \frac{4\pi}{3} J_z \end{bmatrix} \, d\hat{\mathbf{\Omega}} \\
&= \mathbf{J}(\mathbf{x}, E, t)
\end{aligned}$$

que es lo que queríamos demostrar.

□

Volviendo a la evaluación del término de scattering, aprovechando el hecho de que la ecuación 2.33 nos da una forma particular para el flujo angular en función de los dos modos $\ell = 0$ y $\ell = 1$, podemos calcular la fuente de scattering q_s dada por la ecuación 2.32 como

$$\begin{aligned} q_s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \frac{1}{4\pi} \int_0^\infty \Sigma_{s_0}(\mathbf{x}, E' \to E) \cdot \phi(\mathbf{x}, E', t)\, dE' \\ + \frac{3}{4\pi} \int_0^\infty \Sigma_{s_1}(\mathbf{x}, E' \to E) \cdot \left(\mathbf{J}(\mathbf{x}, E', t) \cdot \hat{\mathbf{\Omega}}\right)\, dE' \\ + \sum_{\ell=2}^{\infty} \int_0^\infty \left[\Sigma_{s_\ell}(\mathbf{x}, E' \to E) \sum_{m=-\ell}^{+\ell} \Psi_\ell^m(\mathbf{x}, E', t) \cdot Y_\ell^m(\hat{\mathbf{\Omega}})\right]\, dE' \end{aligned} \tag{2.37}$$

que es una ecuación mucho más útil—desde el punto de vista computacional—que la ecuación 2.25, que da un expresión demasiado general y muy difícil de evaluar. Esto es especialmente importante si podemos despreciar los términos para $\ell > 1$ y suponer a lo más scattering linealmente anisótropo (definición 2.6). En este caso entonces

$$\begin{aligned} q_s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \frac{1}{4\pi} \left\{ \int_0^\infty \Sigma_{s_0}(\mathbf{x}, E' \to E) \cdot \phi(\mathbf{x}, E', t)\, dE' \right. \\ \left. + 3 \cdot \int_0^\infty \Sigma_{s_1}(\mathbf{x}, E' \to E) \cdot \left[\mathbf{J}(\mathbf{x}, E', t) \cdot \hat{\mathbf{\Omega}}\right]\, dE' \right\} \end{aligned} \tag{2.38}$$

Más aún, para el caso particular de scattering isótropo (definición 2.5), q_s se reduce a

$$q_s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \frac{1}{4\pi} \int_0^\infty \Sigma_{s_0}(\mathbf{x}, E' \to E) \cdot \phi(\mathbf{x}, E', t)\, dE' \tag{2.39}$$

Para completar la sección, notamos que dado que la fuente de neutrones debida a fisiones se asume isótropa en el marco de referencia del reactor, su evaluación es similar a esta última ecuación 2.39. En efecto, el término de fisiones de la ecuación de transporte ecuación 2.29 es

$$q_f(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \frac{\chi(E)}{4\pi} \int_0^\infty \int_{4\pi} \nu\Sigma_f(\mathbf{x}, E') \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t)\, d\hat{\mathbf{\Omega}}'\, dE'$$

El coeficiente $\nu\Sigma_f$ no depende de $\hat{\mathbf{\Omega}}'$ por lo que puede salir fuera de la integral sobre 4π. Recordando la definición 2.11 de ϕ resulta

$$\begin{aligned} q_f(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) &= \frac{\chi(E)}{4\pi} \int_0^\infty \nu\Sigma_f(\mathbf{x}, E') \cdot \int_{4\pi} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E', t)\, d\hat{\mathbf{\Omega}}'\, dE' \\ &= \frac{\chi(E)}{4\pi} \int_0^\infty \nu\Sigma_f(\mathbf{x}, E') \cdot \phi(\mathbf{x}, E', t)\, dE' \end{aligned} \tag{2.40}$$

Observación. Ni la ecuación 2.39 ni la ecuación 2.40 dependen de la dirección $\hat{\mathbf{\Omega}}$.

2.3.6. Transporte linealmente anisótropo en estado estacionario

En esta tesis hacemos foco en el caso

1. estacionario, y
2. con scattering linealmente anisótropo (a lo más)

En estas condiciones, la ecuación de transporte queda

$$\begin{aligned} \hat{\mathbf{\Omega}} \cdot \operatorname{grad}\left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E)\right] + \Sigma_t(\mathbf{x}, E) \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E) = \\ \frac{1}{4\pi} \cdot \int_0^\infty \Sigma_{s_0}(\mathbf{x}, E' \to E) \cdot \int_{4\pi} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E')\, d\hat{\mathbf{\Omega}}'\, dE' + \\ \frac{3 \cdot \hat{\mathbf{\Omega}}}{4\pi} \cdot \int_0^\infty \Sigma_{s_1}(\mathbf{x}, E' \to E) \cdot \int_{4\pi} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E') \cdot \hat{\mathbf{\Omega}}'\, d\hat{\mathbf{\Omega}}'\, dE' \\ + \frac{\chi(E)}{4\pi} \int_0^\infty \nu\Sigma_f(\mathbf{x}, E') \cdot \phi(\mathbf{x}, E')\, dE' + s(\mathbf{x}, \hat{\mathbf{\Omega}}, E) \end{aligned} \tag{2.41}$$

Si el scattering es isótropo entonces $\Sigma_{s_1} = 0$ y el segundo término del miembro derecho se anula.

2.3.7. Condiciones iniciales y de contorno

Como ya hemos mencionado en la sección 2.3.4 luego de introducir la ecuación 2.29, la ecuación de transporte es una ecuación integro-diferencial en derivadas parciales de primer orden sobre las coordenadas espaciales $\mathbf{x}$ en un cierto dominio espacial $U \in \mathbb{R}^3$ y una derivada temporal de primer orden sobre $t \in [0, \infty]$. Luego debemos dar condiciones de contorno $\psi(\mathbf{x} = \partial U, E, \hat{\mathbf{\Omega}} = \hat{\mathbf{\Omega}}^*, t)$ sobre la frontera ∂U del dominio U también para todas las energías E y tiempos t pero no para todas las direcciones $\hat{\mathbf{\Omega}}$ sino para un subconjunto $\hat{\mathbf{\Omega}}^*$ ya que la ecuación es de primer orden. Esto es, para cada punto $\mathbf{x} \in \partial U$ sólo se debe fijar el flujo angular ψ correspondiente a las direcciones $\hat{\mathbf{\Omega}}^*$ que *entren* al dominio U, es decir tal que el producto interno $\hat{\mathbf{\Omega}}^* \cdot \hat{\mathbf{n}} < 0$, donde $\hat{\mathbf{n}}$ es el vector normal externo a la frontera ∂U en el punto $\mathbf{x}$. En forma equivalente, se puede pensar como que el flujo angular ψ puede estar fijado, para cada dirección, a lo más en un único punto del espacio ya que la ecuación es de primer grado. Si estuviese fijado en dos puntos, el problema matemático estaría mal definido, como ilustramos en la figura 2.8.

Definición 2.14 (condición de contorno de vacío). Llamamos *condición de contorno de vacío* a la situación en la cual todos los flujos angulares entrantes a U son nulos:

$$\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = 0 \qquad \forall \mathbf{x} \in \Gamma_V \subset \partial U \wedge \hat{\mathbf{\Omega}} \cdot \hat{\mathbf{n}}(\mathbf{x}) < 0$$

Para cada dirección entrante $\hat{\mathbf{\Omega}}/\hat{\mathbf{\Omega}} \cdot \hat{\mathbf{n}} < 0$ definimos el conjunto $\Gamma_V \subset \partial U$ como el lugar geométrico de todos los puntos $\mathbf{x}$ donde imponemos esta condición de contorno.

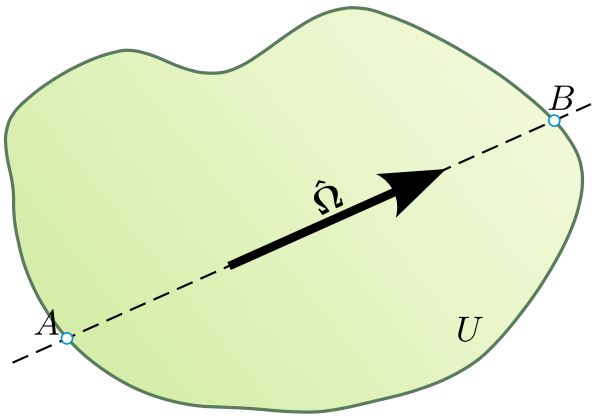


Figura 2.8.: Para una dirección $\hat{\mathbf{\Omega}}$ fija, la ecuación de transporte es una ecuación diferencial de primer orden sobre el espacio. Matemáticamente, esta ecuación puede tener una única condición de contorno o bien en el punto A o bien en el punto B, pero no en ambos. Físicamente, sólo tiene sentido que la condición esté sobre el punto A ya que la dirección $\hat{\mathbf{\Omega}}$ entra al dominio U.

Definición 2.15 (condición de contorno de espejo). Llamamos *condición de contorno de reflexión o de simetría o tipo espejo* cuando el flujo angular entrante en el punto $\mathbf{x} \in \partial U$ es igual al flujo angular saliente en la dirección reflejada

$$\hat{\mathbf{\Omega}}' = \hat{\mathbf{\Omega}} - 2\left(\hat{\mathbf{\Omega}} \cdot \hat{\mathbf{n}}\right)\hat{\mathbf{n}} \tag{2.42}$$

con respecto a la normal exterior $\hat{\mathbf{n}}(\mathbf{x})$ (figura 2.9)

$$\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \psi\left[\mathbf{x}, \hat{\mathbf{\Omega}} - 2\left(\hat{\mathbf{\Omega}} \cdot \hat{\mathbf{n}}\right)\hat{\mathbf{n}}, E, t\right] \qquad \forall \mathbf{x} \in \Gamma_M \subset \partial U \wedge \hat{\mathbf{\Omega}} \cdot \hat{\mathbf{n}}(\mathbf{x}) < 0$$

Definimos el conjunto $\Gamma_M \subset \partial U$ como el lugar geométrico de todos los puntos $\mathbf{x}$ donde imponemos esta condición de contorno.

Las dos condiciones de contorno dadas en la definición 2.14 y en la definición 2.15 son de tipo Dirichlet ya que se fija el valor de la incógnita ψ. Además ambas son homogéneas porque el valor fijado es igual a cero.

Definición 2.16. Si

a. las fuentes de fisión son idénticamente nulas, o
b. las fuentes de fisión son no nulas *y* las fuentes independientes también son no nulas

entonces es posible tener una *condición de contorno general* de Dirichlet no homogénea

$$\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \psi_\Gamma(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \neq 0 \qquad \forall \mathbf{x} \in \Gamma_I \not\subset (\Gamma_V \cup \Gamma_M) \wedge \hat{\mathbf{\Omega}} \cdot \hat{\mathbf{n}}(\mathbf{x}) < 0$$

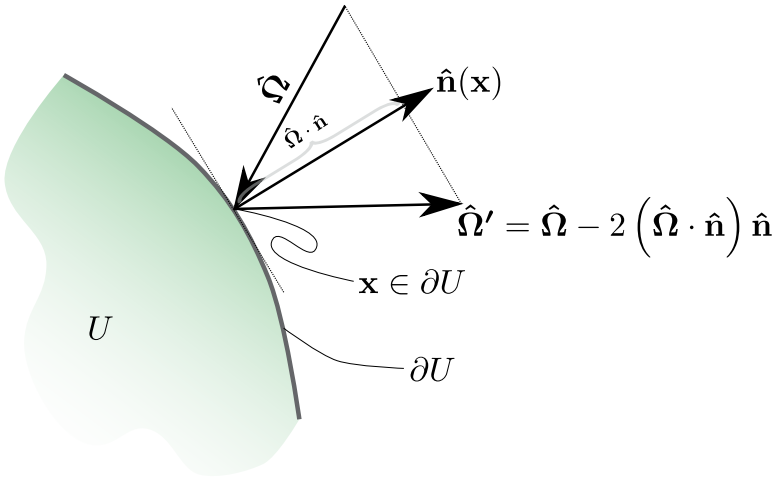


Figura 2.9.: La dirección reflejada $\hat{\mathbf{\Omega}}'$ de la dirección incidente con respecto a la normal exterior $\hat{\mathbf{n}}$ al dominio U en el punto de la frontera $\mathbf{x} \in \partial U$. Se cumple que $\hat{\mathbf{\Omega}} \cdot \hat{\mathbf{n}} = -\hat{\mathbf{\Omega}}' \cdot \hat{\mathbf{n}}$.

Definimos el conjunto $\Gamma_I \subset \partial U$ como el lugar geométrico de todos los puntos $\mathbf{x}$ donde imponemos esta condición de contorno.

Observación. Los problemas estacionarios en los cuales la única fuente de neutrones proviene de fisiones no admiten condiciones de contorno inhomogéneas.

2.4. Aproximación de difusión

La ecuación de difusión de neutrones es una aproximación muy útil que permite

a. obtener soluciones analíticas aproximadas en algunas geometrías simples, y
b. transformar una ecuación diferencial hiperbólica de primer orden en una elíptica de segundo orden sin dependencia angular explícita, simplificando sensiblemente las soluciones numéricas debido a que
    i. la ecuación de difusión discretizada presenta mucho menos grados de libertad que otras formulaciones, tales como ordenadas discretas, y
    ii. la discretización numérica del operador elíptico deviene en matrices simétricas y definidas positivas que permiten la aplicación de algoritmos de resolución muy eficientes, tales como los métodos multi-grid [11], [36].

En esta sección derivamos la ecuación de difusión a partir de la ecuación de transporte. La segunda puede ser considerada *exacta* en el sentido de que todas las deducciones lógicas e igualdades entre miembros han sido estrictas, si damos por cierto las siete suposicionesde la página 30. La primera es una aproximación que, como mostramos en esta sección, proviene de igualar la corriente $\mathbf{J}$ en forma aproximada a un coeficiente negativo por el gradiente $\nabla\phi$ del flujo escalar despreciando la contribución de los términos con $\ell \geq 2$ en la expansión en armónicos esféricos del flujo angular ψ.

2.4.1. Momento de orden cero

Comenzamos integrando la ecuación de transporte de neutrones 2.28 sobre todas los direcciones $\hat{\mathbf{\Omega}}$ para obtener

$$\begin{aligned}\int_{4\pi} \frac{1}{v}\frac{\partial}{\partial t}\Big[\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\Big]\, d\hat{\mathbf{\Omega}} + \int_{4\pi} \hat{\mathbf{\Omega}}\cdot \text{grad}\left[\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\right]\, d\hat{\mathbf{\Omega}} \\ + \int_{4\pi} \Sigma_t(\mathbf{x},E)\cdot\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\, d\hat{\mathbf{\Omega}} = \int_{4\pi} q(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\, d\hat{\mathbf{\Omega}}\end{aligned} \tag{2.43}$$

Teorema 2.9 (extensión de la regla de la derivada del producto). *La divergencia del producto entre el escalar $a(\mathbf{x})$ y el vector $\mathbf{b}(\mathbf{x})$ es*

$$\text{div}\big[a(\mathbf{x})\cdot\mathbf{b}(\mathbf{x})\big] = a(\mathbf{x})\cdot\text{div}\left[\mathbf{b}(\mathbf{x})\right] + \mathbf{b}(\mathbf{x})\cdot\text{grad}\left[a(\mathbf{x})\right]$$

Corolario 2.7. *Para $a=\psi$ y $\mathbf{b}=\hat{\mathbf{\Omega}}$ y el escalar ψ,*

$$\text{div}\left(\hat{\mathbf{\Omega}}\cdot\psi\right) = \hat{\mathbf{\Omega}}\cdot\text{grad}\left(\psi\right) + \psi\cdot\text{div}(\hat{\mathbf{\Omega}})$$

Por la ecuación 2.23 el operador diferencial actúa solamente sobre las coordenadas espaciales. Entonces div$(\hat{\mathbf{\Omega}})=0$. Luego

$$\text{div}\left(\hat{\mathbf{\Omega}}\cdot\psi\right) = \hat{\mathbf{\Omega}}\cdot\text{grad}\left(\psi\right)$$

El segundo término de la ecuación 2.43 queda entonces como la divergencia de la integral sobre $\hat{\mathbf{\Omega}}$ del producto escalar entre la dirección $\hat{\mathbf{\Omega}}$ por el flujo angular ψ

$$\begin{aligned}\frac{1}{v}\frac{\partial}{\partial t}\left[\int \psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\, d\hat{\mathbf{\Omega}}\right] + \text{div}\left[\int\left(\hat{\mathbf{\Omega}}\cdot\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\right)\, d\hat{\mathbf{\Omega}}\right] \\ +\Sigma_t(\mathbf{x},E)\cdot\left[\int \psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\, d\hat{\mathbf{\Omega}}\right] = \int q(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\, d\hat{\mathbf{\Omega}}\end{aligned}$$

Recordando una vez más la definición 2.11 del flujo escalar ϕ y definición 2.12 del vector corriente $\mathbf{J}$,

$$\begin{aligned}\phi(\mathbf{x},E,t) &= \int_{4\pi}\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\, d\hat{\mathbf{\Omega}} \\ \mathbf{J}(\mathbf{x},E,t) &= \int_{4\pi}\left[\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\cdot\hat{\mathbf{\Omega}}\right]\, d\hat{\mathbf{\Omega}}\end{aligned}$$

y definiendo una fuente de neutrones escalar que incluya scattering, fisiones y/o fuentes independientes integrada sobre 4π

$$Q(\mathbf{x}, E, t) = \int_{4\pi} q_s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d\hat{\mathbf{\Omega}} + \int_{4\pi} q_f(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d\hat{\mathbf{\Omega}} + \int_{4\pi} s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d\hat{\mathbf{\Omega}}$$

podemos escribir

$$\frac{1}{v}\frac{\partial}{\partial t}\Big[\phi(\mathbf{x}, E, t)\Big] + \operatorname{div}\Big[\mathbf{J}(\mathbf{x}, E, t)\Big] + \Sigma_t(\mathbf{x}, E) \cdot \phi(\mathbf{x}, E, t) = Q(\mathbf{x}, E, t) \tag{2.44}$$

Observación. La ecuación 2.44 refleja la conservación del momento de orden cero del flujo angular ψ de neutrones con respecto a las direcciones $\hat{\mathbf{\Omega}}$, es decir la conservación de neutrones totales. Dado que proviene de integrar la ecuación de transporte sobre todas las direcciones posible, es tan exacta como la propia ecuación de transporte.

2.4.1.1. Producciones

El miembro derecho de la ecuación 2.44 representa las producciones de neutrones integradas sobre todas las direcciones y es igual a la integral de las tres contribuciones individuales debidas a scattering, fisión y fuentes independientes:

$$\begin{aligned} Q(\mathbf{x}, E, t) &= \int_{4\pi} q_s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d\hat{\mathbf{\Omega}} + \int_{4\pi} q_f(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d\hat{\mathbf{\Omega}} + \int_{4\pi} s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d\hat{\mathbf{\Omega}} \\ &= Q_s(\mathbf{x}, E, t) + Q_f(\mathbf{x}, E, t) + S(\mathbf{x}, E, t) \end{aligned}$$

2.4.1.1.1. Fuente por scattering

Para evaluar la contribución debida al scattering de neutrones podemos integrar la ecuación 2.37 sobre todas las direcciones emergentes $\hat{\mathbf{\Omega}}$:

$$\begin{aligned} Q_s(\mathbf{x}, E, t) = \int_{4\pi} \Bigg\{ & \frac{1}{4\pi} \int_0^\infty \Sigma_{s_0}(\mathbf{x}, E' \to E) \cdot \phi(\mathbf{x}, E', t)\, dE' \\ & + \frac{3}{4\pi} \int_0^\infty \Sigma_{s_1}(\mathbf{x}, E' \to E) \cdot \Big(\mathbf{J}(\mathbf{x}, E', t) \cdot \hat{\mathbf{\Omega}}\Big)\, dE' \\ & + \sum_{\ell=2}^{\infty} \int_0^\infty \left[\Sigma_{s_\ell}(\mathbf{x}, E' \to E) \sum_{m=-\ell}^{+\ell} \Psi_\ell^m(\mathbf{x}, E', t) \cdot Y_\ell^m(\hat{\mathbf{\Omega}})\right] dE' \Bigg\} d\hat{\mathbf{\Omega}} \end{aligned}$$

Por la mismas razones que las esgrimidas al analizar la ecuación 2.34, solamente el primer término del integrando sobre $\hat{\mathbf{\Omega}}$ resulta en una contribución diferente de cero. Luego la fuente de neutrones debido a scattering integrada en 4π se simplifica a

$$Q_s(\mathbf{x}, E, t) = \int_0^\infty \Sigma_{s_0}(\mathbf{x}, E' \to E) \cdot \phi(\mathbf{x}, E', t)\, dE' \tag{2.45}$$

2.4.1.1.2. Fuente por fisión

El término que representa la fuente por fisión es la integral sobre todas las posibles direcciones del término de fuentes de fisión q_f. Para el caso de la ecuación 2.40, tenemos

$$\begin{aligned} Q_f(\mathbf{x}, E, t) &= \int_{4\pi} \left[\frac{\chi(E)}{4\pi} \int_0^\infty \nu\Sigma_f(\mathbf{x}, E') \cdot \phi(\mathbf{x}, E', t)\, dE' \right] d\hat{\mathbf{\Omega}} \\ &= \chi(E) \int_0^\infty \nu\Sigma_f(\mathbf{x}, E') \cdot \phi(\mathbf{x}, E', t)\, dE' \end{aligned} \tag{2.46}$$

2.4.1.1.3. Fuente independiente

La fuente independiente integrada es directamente la integral sobre $\hat{\mathbf{\Omega}}$ de la fuente independiente angular $s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)$:

$$S(\mathbf{x}, E, t) = \int_{4\pi} s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\, d\hat{\mathbf{\Omega}} = s_0(\mathbf{x}, E, t) \tag{2.47}$$

es decir, el momento de orden cero de la distribución angular de la fuente s.

2.4.2. Momento de orden uno

Comencemos esta sección recordando la ecuación 2.28, explicitando los términos de fuentes por scattering (ecuación 2.32), fisión (ecuación 2.26) y fuentes independientes:

$$\begin{aligned} &\frac{1}{v}\frac{\partial}{\partial t}\left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\right] + \hat{\mathbf{\Omega}} \cdot \text{grad}\left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\right] + \Sigma_t(\mathbf{x}, E) \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \\ &\int_0^\infty \sum_{\ell=0}^{\infty} \left[\Sigma_{s_\ell}(\mathbf{x}, E' \to E) \sum_{m=-\ell}^{+\ell} \Psi_\ell^m(\mathbf{x}, E', t) \cdot Y_\ell^m(\hat{\mathbf{\Omega}}) \right] dE' \\ &+ \frac{\chi(E)}{4\pi} \int_0^\infty \nu\Sigma_f(\mathbf{x}, E') \cdot \phi(\mathbf{x}, E', t)\, dE' + s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \end{aligned} \tag{2.48}$$

Multipliquemos esta ecuación escalar por el versor $\hat{\mathbf{\Omega}}$ e integremos sobre todas las posibles direcciones para obtener una ecuación vectorial de dimensión tres:

$$\begin{aligned}
&\underbrace{\int_{4\pi}\left(\frac{1}{v}\frac{\partial}{\partial t}\left[\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\right]\cdot\hat{\mathbf{\Omega}}\right)d\hat{\mathbf{\Omega}}}_{\text{derivada temporal}}+\underbrace{\int_{4\pi}\left(\hat{\mathbf{\Omega}}\cdot\operatorname{grad}\left[\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\right]\cdot\hat{\mathbf{\Omega}}\right)d\hat{\mathbf{\Omega}}}_{\text{advección}}\\
&\qquad+\underbrace{\int_{4\pi}\left(\Sigma_t(\mathbf{x},E)\cdot\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\cdot\hat{\mathbf{\Omega}}\right)d\hat{\mathbf{\Omega}}}_{\text{absorción total}}=\\
&\underbrace{\int_{4\pi}\left(\int_0^\infty\sum_{\ell=0}^{\infty}\left[\Sigma_{s_\ell}(\mathbf{x},E'\to E)\sum_{m=-\ell}^{+\ell}\Psi_\ell^m(\mathbf{x},E',t)\cdot Y_\ell^m(\hat{\mathbf{\Omega}})\right]dE'\cdot\hat{\mathbf{\Omega}}\right)d\hat{\mathbf{\Omega}}}_{\text{scattering}}\\
&+\underbrace{\int_{4\pi}\left(\frac{\chi(E)}{4\pi}\int_0^\infty\nu\Sigma_f(\mathbf{x},E')\cdot\phi(\mathbf{x},E',t)\,dE'\cdot\hat{\mathbf{\Omega}}\right)d\hat{\mathbf{\Omega}}}_{\text{fisión}}+\underbrace{\int_{4\pi}\left(s(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\cdot\hat{\mathbf{\Omega}}\right)d\hat{\mathbf{\Omega}}}_{\text{fuentes independientes}}
\end{aligned}\tag{2.49}$$

Analicemos en las seis sub-secciones que siguen los términos de esta expresión un por uno, teniendo en cuenta los desarrollos matemáticos que hemos realizado a lo largo de todo el capítulo.

2.4.2.1. Derivada temporal

El primer término corresponde a la derivada temporal de la corriente. En efecto

$$\begin{aligned}
\int_{4\pi}\left(\frac{1}{v}\frac{\partial}{\partial t}\left[\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\right]\cdot\hat{\mathbf{\Omega}}\right)d\hat{\mathbf{\Omega}}&=\frac{1}{v(E)}\frac{\partial}{\partial t}\left[\int_{4\pi}\left(\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\cdot\hat{\mathbf{\Omega}}\right)d\hat{\mathbf{\Omega}}\right]\\
&=\sqrt{\frac{m}{2E}}\frac{\partial}{\partial t}\Big[\mathbf{J}(\mathbf{x},E,t)\Big]
\end{aligned}\tag{2.50}$$

2.4.2.2. Advección

El término de advección parece el más sencillo pero de hecho es el más difícil de analizar. Es más, es tan complicado que es aquí donde necesitamos hacer la aproximación más importante de la formulación de difusión. Tenemos que suponer que los coeficientes Ψ_ℓ^m de la expansión del flujo angular ψ en armónicos esféricos dado en la ecuación 2.33 son cero para $\ell \geq 2$, es decir

$$\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\simeq\frac{1}{4\pi}\left[\phi(\mathbf{x},E,t)+3\cdot\Big(\mathbf{J}(\mathbf{x},E,t)\cdot\hat{\mathbf{\Omega}}\Big)\right]$$

Con esta suposición, el término de advección queda aproximadamente igual a

$$\begin{aligned}
&\int_{4\pi}\left(\hat{\mathbf{\Omega}}\cdot\operatorname{grad}\left[\psi(\mathbf{x},\hat{\mathbf{\Omega}},E,t)\right]\cdot\hat{\mathbf{\Omega}}\right)d\hat{\mathbf{\Omega}}\simeq\\
&\frac{1}{4\pi}\cdot\left\{\int_{4\pi}\Big(\hat{\mathbf{\Omega}}\cdot\operatorname{grad}(\phi)\Big)\cdot\hat{\mathbf{\Omega}}\,d\hat{\mathbf{\Omega}}+3\cdot\int_{4\pi}\Big[\hat{\mathbf{\Omega}}\cdot\operatorname{grad}\Big(\mathbf{J}\cdot\hat{\mathbf{\Omega}}\Big)\Big]\cdot\hat{\mathbf{\Omega}}\,d\hat{\mathbf{\Omega}}\right\}
\end{aligned}\tag{2.51}$$

La integral sobre el gradiente flujo escalar $\nabla\phi$ es

$$\begin{aligned}
\int_{4\pi}\left[\hat{\mathbf{\Omega}}\cdot\operatorname{grad}(\phi)\right]\cdot\hat{\mathbf{\Omega}}\,d\hat{\mathbf{\Omega}} &= \int_{4\pi}\left(\begin{bmatrix}\hat{\Omega}_x & \hat{\Omega}_y & \hat{\Omega}_z\end{bmatrix}\begin{bmatrix}\frac{\partial\phi}{\partial x}\\ \frac{\partial\phi}{\partial y}\\ \frac{\partial\phi}{\partial z}\end{bmatrix}\right)\cdot\begin{bmatrix}\hat{\Omega}_x\\ \hat{\Omega}_y\\ \hat{\Omega}_z\end{bmatrix}d\hat{\mathbf{\Omega}}\\
&= \int_{4\pi}\left(\hat{\Omega}_x\cdot\frac{\partial\phi}{\partial x}+\hat{\Omega}_y\cdot\frac{\partial\phi}{\partial y}+\hat{\Omega}_z\cdot\frac{\partial\phi}{\partial z}\right)\cdot\begin{bmatrix}\hat{\Omega}_x\\ \hat{\Omega}_y\\ \hat{\Omega}_z\end{bmatrix}d\hat{\mathbf{\Omega}}\\
&= \int_{4\pi}\begin{bmatrix}\hat{\Omega}_x\hat{\Omega}_x\cdot\frac{\partial\phi}{\partial x}+\hat{\Omega}_x\hat{\Omega}_y\cdot\frac{\partial\phi}{\partial y}+\hat{\Omega}_x\hat{\Omega}_z\cdot\frac{\partial\phi}{\partial z}\\ \hat{\Omega}_y\hat{\Omega}_x\cdot\frac{\partial\phi}{\partial x}+\hat{\Omega}_y\hat{\Omega}_y\cdot\frac{\partial\phi}{\partial y}+\hat{\Omega}_y\hat{\Omega}_z\cdot\frac{\partial\phi}{\partial z}\\ \hat{\Omega}_z\hat{\Omega}_x\cdot\frac{\partial\phi}{\partial x}+\hat{\Omega}_z\hat{\Omega}_y\cdot\frac{\partial\phi}{\partial y}+\hat{\Omega}_z\hat{\Omega}_z\cdot\frac{\partial\phi}{\partial z}\end{bmatrix}d\hat{\mathbf{\Omega}}\\
&= \left(\int_{4\pi}\begin{bmatrix}\hat{\Omega}_x^2 & \hat{\Omega}_x\hat{\Omega}_y & \hat{\Omega}_x\hat{\Omega}_z\\ \hat{\Omega}_y\hat{\Omega}_x & \hat{\Omega}_y^2 & \hat{\Omega}_y\hat{\Omega}_z\\ \hat{\Omega}_z\hat{\Omega}_x & \hat{\Omega}_z\hat{\Omega}_y & \hat{\Omega}_z^2\end{bmatrix}d\hat{\mathbf{\Omega}}\right)\cdot\operatorname{grad}\left[\phi(\mathbf{x},E,t)\right]
\end{aligned}$$

Recordando el teorema 2.7, los elementos de la diagonal dan como resultado $4\pi/3$ mientras que el resto se anulan:

$$\begin{aligned}
\int_{4\pi}\left[\hat{\mathbf{\Omega}}\cdot\operatorname{grad}(\phi)\right]\cdot\hat{\mathbf{\Omega}}\,d\hat{\mathbf{\Omega}} &= \frac{4\pi}{3}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}\frac{\partial\phi}{\partial x}\\ \frac{\partial\phi}{\partial y}\\ \frac{\partial\phi}{\partial z}\end{bmatrix}\\
&= \frac{4\pi}{3}\cdot\begin{bmatrix}\frac{\partial\phi}{\partial x}\\ \frac{\partial\phi}{\partial y}\\ \frac{\partial\phi}{\partial z}\end{bmatrix} = 4\pi\cdot\left[\frac{1}{3}\cdot\operatorname{grad}\left[\phi(\mathbf{x},E,t)\right]\right]
\end{aligned}$$

Por otro lado, el segundo término de la ecuación 2.51 se anula. En efecto,

$$\int_{4\pi}\left[\hat{\mathbf{\Omega}}\cdot\operatorname{grad}\left(\mathbf{J}\cdot\hat{\mathbf{\Omega}}\right)\right]\cdot\hat{\mathbf{\Omega}}\,d\hat{\mathbf{\Omega}}$$

$$= \int_{4\pi}\left\{\begin{bmatrix}\hat{\Omega}_x & \hat{\Omega}_y & \hat{\Omega}_z\end{bmatrix}\cdot\operatorname{grad}\left(J_x\cdot\hat{\Omega}_x+J_y\cdot\hat{\Omega}_y+J_z\cdot\hat{\Omega}_z\right)\right\}\cdot\begin{bmatrix}\hat{\Omega}_x\\ \hat{\Omega}_y\\ \hat{\Omega}_z\end{bmatrix}d\hat{\mathbf{\Omega}}$$

$$= \int_{4\pi}\left\{\begin{bmatrix}\hat{\Omega}_x & \hat{\Omega}_y & \hat{\Omega}_z\end{bmatrix}\cdot\begin{bmatrix}\frac{\partial J_x}{\partial x}\cdot\hat{\Omega}_x+\frac{\partial J_y}{\partial x}\cdot\hat{\Omega}_y+\frac{\partial J_z}{\partial x}\cdot\hat{\Omega}_z\\ \frac{\partial J_x}{\partial y}\cdot\hat{\Omega}_x+\frac{\partial J_y}{\partial y}\cdot\hat{\Omega}_y+\frac{\partial J_z}{\partial y}\cdot\hat{\Omega}_z\\ \frac{\partial J_x}{\partial z}\cdot\hat{\Omega}_x+\frac{\partial J_y}{\partial z}\cdot\hat{\Omega}_y+\frac{\partial J_z}{\partial z}\cdot\hat{\Omega}_z\end{bmatrix}\right\}\cdot\begin{bmatrix}\hat{\Omega}_x\\ \hat{\Omega}_y\\ \hat{\Omega}_z\end{bmatrix}d\hat{\mathbf{\Omega}}$$

$$
\begin{aligned}
= \int_{4\pi} \Bigg(& \frac{\partial J_x}{\partial x} \cdot \hat{\Omega}_x^2 + \frac{\partial J_y}{\partial x} \cdot \hat{\Omega}_y \hat{\Omega}_x + \frac{\partial J_z}{\partial x} \cdot \hat{\Omega}_z \hat{\Omega}_x + \\
& \frac{\partial J_x}{\partial y} \cdot \hat{\Omega}_x \hat{\Omega}_y + \frac{\partial J_y}{\partial y} \cdot \hat{\Omega}_y^2 + \frac{\partial J_z}{\partial y} \cdot \hat{\Omega}_z \hat{\Omega}_y + \\
& \frac{\partial J_x}{\partial z} \cdot \hat{\Omega}_x \hat{\Omega}_z + \frac{\partial J_y}{\partial z} \cdot \hat{\Omega}_y \hat{\Omega}_z + \frac{\partial J_z}{\partial z} \cdot \hat{\Omega}_z^2 \Bigg) \cdot \begin{bmatrix} \hat{\Omega}_x \\ \hat{\Omega}_y \\ \hat{\Omega}_z \end{bmatrix} \, d\hat{\mathbf{\Omega}}
\end{aligned}
$$

El integrando es un vector $\mathbf{v} \in \mathbb{R}^3$ cuyos tres elementos son

$$
\begin{aligned}
v_1 = & \frac{\partial J_x}{\partial x} \cdot \hat{\Omega}_x^3 + \frac{\partial J_y}{\partial x} \cdot \hat{\Omega}_y \hat{\Omega}_x^2 + \frac{\partial J_z}{\partial x} \cdot \hat{\Omega}_z \hat{\Omega}_x^2 + \\
& \frac{\partial J_x}{\partial y} \cdot \hat{\Omega}_x^2 \hat{\Omega}_y + \frac{\partial J_y}{\partial y} \cdot \hat{\Omega}_y^2 \hat{\Omega}_x + \frac{\partial J_z}{\partial y} \cdot \hat{\Omega}_z \hat{\Omega}_y \hat{\Omega}_x + \\
& \frac{\partial J_x}{\partial z} \cdot \hat{\Omega}_x^2 \hat{\Omega}_z + \frac{\partial J_y}{\partial z} \cdot \hat{\Omega}_y \hat{\Omega}_z \hat{\Omega}_x + \frac{\partial J_z}{\partial z} \cdot \hat{\Omega}_z^2 \hat{\Omega}_x
\end{aligned}
$$

$$
\begin{aligned}
v_2 = & \frac{\partial J_x}{\partial x} \cdot \hat{\Omega}_x^2 \hat{\Omega}_y + \frac{\partial J_y}{\partial x} \cdot \hat{\Omega}_y^2 \hat{\Omega}_x + \frac{\partial J_z}{\partial x} \cdot \hat{\Omega}_z \hat{\Omega}_x \hat{\Omega}_y + \\
& \frac{\partial J_x}{\partial y} \cdot \hat{\Omega}_x \hat{\Omega}_y^2 + \frac{\partial J_y}{\partial y} \cdot \hat{\Omega}_y^3 + \frac{\partial J_z}{\partial y} \cdot \hat{\Omega}_z \hat{\Omega}_y^2 + \\
& \frac{\partial J_x}{\partial z} \cdot \hat{\Omega}_x \hat{\Omega}_z \hat{\Omega}_y + \frac{\partial J_y}{\partial z} \cdot \hat{\Omega}_y^2 \hat{\Omega}_z + \frac{\partial J_z}{\partial z} \cdot \hat{\Omega}_z^2 \hat{\Omega}_y
\end{aligned}
$$

y

$$
\begin{aligned}
v_3 = & \frac{\partial J_x}{\partial x} \cdot \hat{\Omega}_x^2 \hat{\Omega}_z + \frac{\partial J_y}{\partial x} \cdot \hat{\Omega}_y \hat{\Omega}_x \hat{\Omega}_z + \frac{\partial J_z}{\partial x} \cdot \hat{\Omega}_z^2 \hat{\Omega}_x + \\
& \frac{\partial J_x}{\partial y} \cdot \hat{\Omega}_x \hat{\Omega}_y \hat{\Omega}_z + \frac{\partial J_y}{\partial y} \cdot \hat{\Omega}_y^2 \hat{\Omega}_z + \frac{\partial J_z}{\partial y} \cdot \hat{\Omega}_z^2 \hat{\Omega}_y + \\
& \frac{\partial J_x}{\partial z} \cdot \hat{\Omega}_x \hat{\Omega}_z^2 + \frac{\partial J_y}{\partial z} \cdot \hat{\Omega}_y \hat{\Omega}_z^2 + \frac{\partial J_z}{\partial z} \cdot \hat{\Omega}_z^3
\end{aligned}
$$

Los veintisiete términos tienen al menos un coseno dirección elevado a un potencia impar, por lo que por el teorema 2.8 su integral sobre 4π es igual a cero. Entonces, podemos aproximar el término de advección bajo la suposición de que el flujo angular es linealmente anisótropo como

$$
\int_{4\pi} \left(\hat{\mathbf{\Omega}} \cdot \operatorname{grad} \left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \right] \cdot \hat{\mathbf{\Omega}} \right) \, d\hat{\mathbf{\Omega}} \simeq \frac{1}{3} \cdot \operatorname{grad} \left[\phi(\mathbf{x}, E, t) \right] \tag{2.52}
$$

Observación. La referencia [10, p. 88] indica

> If one keeps the higher terms in the Legendre polynomial expansion of ψ, there then results, as can be shown by an easy calculation, the exact expression
>
> $$J = -\frac{1}{3\left(\Sigma_{tr} + \Sigma_a\right)} \frac{d}{dx}\left[\phi(x)\left(1 + 2\frac{\psi_2(x)}{\phi(x)}\right)\right]$$

2.4.2.3. Absorción total

El siguiente es el término de absorciones totales escrito en forma vectorial con respecto a la corriente

$$\begin{aligned}\int_{4\pi}\left(\Sigma_t(\mathbf{x}, E)\cdot\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\cdot\hat{\mathbf{\Omega}}\right) d\hat{\mathbf{\Omega}} &= \Sigma_t(\mathbf{x}, E)\cdot\int_{4\pi}\left(\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\cdot\hat{\mathbf{\Omega}}\right) d\hat{\mathbf{\Omega}} \\ &= \Sigma_t(\mathbf{x}, E)\cdot\mathbf{J}(\mathbf{x}, E, t)\end{aligned} \tag{2.53}$$

2.4.2.4. Scattering

El término de scattering parece complicado, pero en realidad ya lo hemos resuelto al derivar la ecuación ecuación 2.35. En primer lugar, partamos de la ecuación ecuación 2.37 multiplicada por $\hat{\mathbf{\Omega}}$ e integrada en 4π:

$$\begin{aligned}&\int_{4\pi}\left[\frac{1}{4\pi}\int_0^\infty \Sigma_{s_0}(\mathbf{x}, E'\to E)\cdot\phi(\mathbf{x}, E', t)\cdot\hat{\mathbf{\Omega}}\, dE'\right] d\hat{\mathbf{\Omega}} \\ &+\int_{4\pi}\left[\frac{3}{4\pi}\int_0^\infty \Sigma_{s_1}(\mathbf{x}, E'\to E)\cdot\left(\mathbf{J}(\mathbf{x}, E', t)\cdot\hat{\mathbf{\Omega}}\right)\cdot\hat{\mathbf{\Omega}}\, dE'\right] d\hat{\mathbf{\Omega}} \\ &+\int_{4\pi}\left\{\sum_{\ell=2}^\infty\int_0^\infty\left[\Sigma_{s_\ell}(\mathbf{x}, E'\to E)\sum_{m=-\ell}^{+\ell}\Psi_\ell^m(\mathbf{x}, E', t)\cdot Y_\ell^m(\hat{\mathbf{\Omega}})\cdot\hat{\mathbf{\Omega}}\right] dE'\right\} d\hat{\mathbf{\Omega}}\end{aligned}$$

En forma similar al argumento planteado en la ecuación 2.34, el primer término se anula. Los términos de la sumatoria para $\ell \geq 2$ también se anulan por la propiedad de ortogonalidad de los armónicos esféricos (teorema 2.4) y la ecuación 2.36 que indica que $\hat{\mathbf{\Omega}}$ es proporcional a $Y_1^m(\hat{\mathbf{\Omega}})$. Entonces el término de scattering queda

$$\begin{aligned}&= \frac{3}{4\pi}\int_{4\pi}\left\{\int_0^\infty \Sigma_{s_1}(\mathbf{x}, E'\to E)\cdot\left(J_x\hat{\Omega}_x + J_y\hat{\Omega}_y + J_z\hat{\Omega}_z\right)\cdot\begin{bmatrix}\hat{\Omega}_x\\ \hat{\Omega}_y\\ \hat{\Omega}_z\end{bmatrix} dE'\right\} d\hat{\mathbf{\Omega}} \\ &= \frac{3}{4\pi}\int_{4\pi}\left\{\int_0^\infty \Sigma_{s_1}(\mathbf{x}, E'\to E)\cdot\begin{bmatrix}J_x\hat{\Omega}_x\hat{\Omega}_x + J_y\hat{\Omega}_y\hat{\Omega}_x + J_z\hat{\Omega}_z\hat{\Omega}_x\\ J_x\hat{\Omega}_x\hat{\Omega}_y + J_y\hat{\Omega}_y\hat{\Omega}_y + J_z\hat{\Omega}_z\hat{\Omega}_y\\ J_x\hat{\Omega}_x\hat{\Omega}_z + J_y\hat{\Omega}_y\hat{\Omega}_z + J_z\hat{\Omega}_z\hat{\Omega}_z\end{bmatrix} dE'\right\} d\hat{\mathbf{\Omega}}\end{aligned}$$

Teniendo además en cuenta el teorema 2.7, el término de scattering toma la inocua forma de

$$\int_0^\infty \Sigma_{s_1}(\mathbf{x}, E' \to E) \cdot \mathbf{J}(\mathbf{x}, E', t)\, dE' \tag{2.54}$$

2.4.2.5. Fisiones

El siguiente es el término de fisiones, cuya integral se anula. En efecto,

$$\int_{4\pi} \left(\frac{\chi(E)}{4\pi} \int_0^\infty \nu\Sigma_f(\mathbf{x}, E') \cdot \phi(\mathbf{x}, E', t)\, dE' \cdot \hat{\mathbf{\Omega}} \right) d\hat{\mathbf{\Omega}} = 0 \tag{2.55}$$

ya que el integrando es el producto de una factor que no depende del ángulo por el versor $\hat{\mathbf{\Omega}}$. En forma equivalente, estamos evaluando el momento de orden $\ell = 1$ de una fuente de fisión. Dado que ésta es isótrópa, el único momento diferente de cero es el de orden $\ell = 0$.

2.4.2.6. Fuentes independientes

El término de fuentes independientes es igual a un vector cuyas componentes son los tres coeficientes correspondientes a $\ell = 1$ en la expansión en armónicos esféricos sobre el ángulo $\hat{\mathbf{\Omega}}$ de la fuente s:

$$\begin{aligned} \int_{4\pi} s(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) \cdot \hat{\mathbf{\Omega}}\, d\hat{\mathbf{\Omega}} &= \sqrt{\frac{3}{4\pi}} \cdot \begin{bmatrix} s_1^{+1}(\mathbf{x}, E, t) \\ s_1^{-1}(\mathbf{x}, E, t) \\ s_1^{0}(\mathbf{x}, E, t) \end{bmatrix} \\ &= \sqrt{\frac{3}{4\pi}} \cdot \mathbf{s}_1(\mathbf{x}, E, t) \end{aligned} \tag{2.56}$$

Si las fuentes independientes son isotrópicas en el marco de referencia del reactor, los tres coeficientes son cero y la integral es nula.

2.4.3. Ley de Fick

Estamos entonces en condiciones de volver a reunir los seis términos de la ecuación 2.49 que analizamos por separado en las ecuaciones

- 2.50 (derivada temporal),
- 2.52 (advección),
- 2.53 (absorciones totales),
- 2.54 (scattering),
- 2.55 (fisiones), y
- 2.56 (fuentes)

y concluir que al multiplicar la ecuación 2.48 por $\hat{\mathbf{\Omega}}$ e integrar en todas las posibles direcciones, obtenemos

$$\begin{aligned}\frac{1}{v(E)}\frac{\partial \mathbf{J}}{\partial t} + \frac{1}{3}\cdot \text{grad}\big[\phi(\mathbf{x},E,t)\big] + \Sigma_t(\mathbf{x},E)\cdot\mathbf{J}(\mathbf{x},E,t) = \\ \int_0^\infty \Sigma_{s_1}(\mathbf{x},E'\to E)\cdot\mathbf{J}(\mathbf{x},E',t)\,dE' + \sqrt{\frac{3}{4\pi}}\cdot\mathbf{s}_1(\mathbf{x},E,t)\end{aligned} \tag{2.57}$$

Para arribar finalmente a la ecuación de difusión necesitamos tres nuevas suposiciones:

i. Que

   a. el problema sea estacionario, o

   b. que

$$\frac{3}{v}\frac{\partial \mathbf{J}}{\partial t} \ll \text{grad}\,[\phi(\mathbf{x},E,t)]$$

ii. Que

   a. no haya fuentes independientes, o

   b. que la fuente independiente sea isótropa de forma tal que los tres coeficientes s_ℓ^m de la ecuación 2.55 sean iguales a cero.

iii. Que

   a. el scattering sea isótropo (en el marco de referencia del reactor), o

   b. que

$$\int_0^\infty \Sigma_{s_1}(\mathbf{x},E'\to E)\cdot\mathbf{J}(\mathbf{x},E',t)\,dE' \simeq \int_0^\infty \Sigma_{s_1}(\mathbf{x},E\to E')\cdot\mathbf{J}(\mathbf{x},E,t)\,dE' \tag{2.58}$$

   El miembro izquierdo representa el in-scattering de neutrones de todas las energías mientras que el miembro derecho es el out-scattering de neutrones de energía E hacia todas las otras energías E'. Si la absorción es pequeña, estas dos expresiones se deberían balancear aproximadamente.

Si al menos una de las dos condiciones a ó b de cada una de las tres suposiciones i, ii y iii son ciertas (o razonables), entonces volviendo a la ecuación 2.57, tenemos

$$\begin{aligned}\frac{1}{3}\cdot\operatorname{grad}\left[\phi(\mathbf{x},E,t)\right]+\Sigma_t(\mathbf{x},E)\cdot\mathbf{J}(\mathbf{x},E,t) &\simeq \int_0^\infty \Sigma_{s_1}(\mathbf{x},E\to E')\cdot\mathbf{J}(\mathbf{x},E,t)\,dE' \\ &\simeq \int_0^\infty \Sigma_{s_1}(\mathbf{x},E\to E')\,dE'\cdot\mathbf{J}(\mathbf{x},E,t) \\ &\simeq \mu_0(\mathbf{x},E)\int_0^\infty \Sigma_{s_0}(\mathbf{x},E\to E')\,dE'\cdot\mathbf{J}(\mathbf{x},E,t) \\ &\simeq \mu_0(\mathbf{x},E)\cdot\Sigma_{s_t}(\mathbf{x},E)\cdot\mathbf{J}(\mathbf{x},E,t)\end{aligned}$$

donde en los últimos dos pasos hemos utilizado la definición 2.7 y la sección eficaz *total* de scattering $\Sigma_s(\mathbf{x},E)$ de la ecuación 2.2 evaluada en función del coeficiente Σ_{s_0} según la ecuación 2.5. Con esta expresión podemos relacionar el vector corriente $\mathbf{J}$ con el gradiente del flujo escalar ϕ como

$$\mathbf{J}(\mathbf{x},E,t)=-\frac{1}{3\left[\Sigma_t(\mathbf{x},E)-\mu_0(\mathbf{x},E)\cdot\Sigma_{s_t}(\mathbf{x},E)\right]}\cdot\operatorname{grad}\left[\phi(\mathbf{x},E,t)\right]$$

Definición 2.17. El *coeficiente de difusión* D definido como

$$D(\mathbf{x},E)=\frac{1}{3\left[\Sigma_t(\mathbf{x},E)-\mu_0(\mathbf{x},E)\cdot\Sigma_{s_t}(\mathbf{x},E)\right]} \tag{2.59}$$

es tal que, si

1. el flujo angular es linealmente anisótropo $\psi\simeq(\phi+3\mathbf{J})/4\pi$
2. el problema es estacionario o $3/v\cdot\partial\mathbf{J}/\partial t\ll\nabla\phi$
3. no hay fuentes independientes o éstas son isotrópicas
4. el scattering es isótropo o el in-scattering es similar al out-scattering $\int\Sigma_{s_1}(E'\to E)\cdot\mathbf{J}(E')\,dE'\simeq\int\Sigma_{s_1}(E\to E')\cdot\mathbf{J}(E)\,dE'$

entonces se cumple la *Ley de Fick*

$$\mathbf{J}(\mathbf{x},E,t)\simeq -D(\mathbf{x},E)\cdot\operatorname{grad}\left[\phi(\mathbf{x},E,t)\right] \tag{2.60}$$

según la cual el vector corriente $\mathbf{J}$ es proporcional a menos el gradiente del flujo escalar ϕ a través de un coeficiente de difusión D dado por la ecuación 2.59.

Observación. La ley de Fick refleja, en forma aproximada, la conservación del momento de orden uno del flujo angular ψ con respecto a todas las direcciones de vuelo $\hat{\mathbf{\Omega}}$.

Observación. Dado que las secciones eficaces macroscópicas tienen unidades de inversa de longitud (definición 2.1), el coeficiente de difusión D tiene unidades de longitud.

2.4.4. La ecuación de difusión

Podemos combinar los dos resultados de la conservación de momentos de orden cero y uno desarrollados en las secciones anteriores teniendo en cuenta las expresiones dadas por las ecuaciones

- 2.44 (conservación),
- 2.45 (scattering),
- 2.46 (fisión),
- 2.47 (fuentes), y
- 2.60 (ley de Fick)

para obtener finalmente la celebrada *ecuación de difusión de neutrones*

$$\sqrt{\frac{m}{2E}}\frac{\partial}{\partial t}\Big[\phi(\mathbf{x},E,t)\Big] - \operatorname{div}\Big[D(\mathbf{x},E)\cdot\operatorname{grad}\left[\phi(\mathbf{x},E,t)\right]\Big] + \Sigma_t(\mathbf{x},E)\cdot\phi(\mathbf{x},E,t) = \int_0^\infty \Sigma_{s_0}(\mathbf{x},E'\to E)\cdot\phi(\mathbf{x},E',t)\,dE' + \chi(E)\int_0^\infty \nu\Sigma_f(\mathbf{x},E')\cdot\phi(\mathbf{x},E',t)\,dE' + s_0(\mathbf{x},E,t) \tag{2.61}$$

que es una ecuación integro-diferencial elíptica en derivadas parciales de segundo orden sobre el espacio (los operadores divergencia y gradiente operan sólo sobre las coordenadas espaciales), y de primer orden sobre el tiempo para la incógnita ϕ definida sobre

1. el espacio $\mathbf{x}$,
2. la energía E, y
3. el tiempo t.

Observación. La incógnita de la ecuación de difusión es el flujo escalar ϕ que no depende de $\hat{\mathbf{\Omega}}$.

Los datos de entrada para la ecuación de difusión de neutrones son:

- Las secciones eficaces Σ_t y $\nu\Sigma_f$ como función del espacio $\mathbf{x}$ y de la energía E.
- El espectro de fisión χ en función de la energía E.
- El coeficiente de difusión D como función del espacio $\mathbf{x}$ y de la energía E.
- La fuente independiente de neutrones s, que debe ser isótropa.
- El parámetro constante m, que es la masa en reposo del neutrón.

Observación. En estado estacionario, la ecuación de difusión de neutrones es

$$-\operatorname{div}\Big[D(\mathbf{x},E)\cdot\operatorname{grad}\left[\phi(\mathbf{x},E)\right]\Big] + \Sigma_t(\mathbf{x},E)\cdot\phi(\mathbf{x},E) = \int_0^\infty \Sigma_{s_0}(\mathbf{x},E'\to E)\cdot\phi(\mathbf{x},E')\,dE' + \chi(E)\int_0^\infty \nu\Sigma_f(\mathbf{x},E')\cdot\phi(\mathbf{x},E')\,dE' + s_0(\mathbf{x},E) \tag{2.62}$$

2.4.5. Condiciones de contorno

La ecuación de difusión es elíptica sobre las coordenadas espaciales por lo que debemos especificar, además de las condiciones iniciales apropiadas en el caso transitorio, condiciones de contorno en toda la frontera ∂U del dominio espacial. Para el caso de una ecuación elíptica, éstas pueden ser de tipo

a. Dirichlet donde especificamos el valor del flujo escalar ϕ en $\Gamma_D \subset \partial U$; o
b. Neumann donde fijamos el valor de la derivada normal $\partial\phi/\partial n$ en $\Gamma_N \subset \partial U$; o
c. Robin donde damos una combinación lineal del flujo y de la derivada normal en $\Gamma_R \subset \partial U$.

Observación. Debe cumplirse que $\Gamma_D \cap \Gamma_N \cap \Gamma_R = \emptyset$ y que $\Gamma_D \cup \Gamma_N \cup \Gamma_R = \partial U$.

Dada una superficie diferencial dA cuya normal exterior es $\hat{\mathbf{n}}$, la tasa de neutrones entrante por unidad de área a través de dicha superficie dA está dada por la ecuación 2.19 de la definición 2.12:

$$J_n^-(\mathbf{x}, E, t) = \int_{\hat{\mathbf{\Omega}}\cdot\hat{\mathbf{n}}<0} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\left(\hat{\mathbf{\Omega}}\cdot\hat{\mathbf{n}}\right) d\hat{\mathbf{\Omega}} \tag{2.19}$$

Despreciando los términos para $\ell \geq 2$ de la expansión de la ecuación 2.33 podemos estimar esta corriente como

$$\begin{aligned}
J_n^-(\mathbf{x}, E, t) &\simeq \int_{\hat{\mathbf{\Omega}}\cdot\hat{\mathbf{n}}<0} \frac{1}{4\pi}\left[\phi(\mathbf{x}, E, t) + 3\,\mathbf{J}(\mathbf{x}, E, t)\cdot\hat{\mathbf{\Omega}}\right]\cdot\left(\hat{\mathbf{\Omega}}\cdot\hat{\mathbf{n}}\right) d\hat{\mathbf{\Omega}} \\
&\simeq \frac{1}{4\pi}\phi(\mathbf{x}, E, t)\int_0^{-1} \mu' \cdot 2\pi\, d\mu' + \frac{3}{4\pi}\int_{\hat{\mathbf{\Omega}}\cdot\hat{\mathbf{n}}<0}\left[\mathbf{J}(\mathbf{x}, E, t)\cdot\hat{\mathbf{\Omega}}\right]\cdot\left(\hat{\mathbf{\Omega}}\cdot\hat{\mathbf{n}}\right) d\hat{\mathbf{\Omega}} \\
&\simeq \frac{1}{4\pi}\phi(\mathbf{x}, E, t)\cdot 2\pi\cdot\frac{1}{2} + \frac{3}{4\pi}\cdot\left[\mathbf{J}(\mathbf{x}, E, t)\cdot\hat{\mathbf{n}}\right]\cdot\left(-\frac{2}{3}\pi\right) \\
&\simeq \frac{1}{4}\phi(\mathbf{x}, E, t) - \frac{1}{2}\left[\mathbf{J}(\mathbf{x}, E, t)\cdot\hat{\mathbf{n}}\right]
\end{aligned}$$

donde hemos usado el teorema 2.7 sobre una semi-esfera unitaria para resolver la segunda integral del miembro derecho. A la luz de la ley de Fick dada por la definición 2.17, podemos escribir

$$\begin{aligned}
J_n^-(\mathbf{x}, E, t) &\simeq \frac{1}{4}\cdot\phi(\mathbf{x}, E, t) + \frac{1}{2}\cdot D(\mathbf{x}, E)\cdot\left\{\mathrm{grad}\left[\phi(\mathbf{x}, E, t)\right]\cdot\hat{\mathbf{n}}\right\} \\
&\simeq \frac{\phi(\mathbf{x}, E, t)}{4} + \frac{D(\mathbf{x}, E)}{2}\cdot\frac{\partial\phi}{\partial n}
\end{aligned}$$

lo que nos da una expresión para definir la condición de contorno de vacío para la ecuación de difusión.

Definición 2.18. En forma análoga a la definición 2.14, llamamos *condición de contorno de vacío* a la situación en la cual la corriente entrante a través de una porción de la frontera ∂U es cero, con lo que se debe cumplir:

$$\phi(\mathbf{x}, E, t) + 2\cdot D(\mathbf{x}, E)\cdot\frac{\partial\phi}{\partial n} = 0 \qquad \forall\mathbf{x} \in \Gamma_V \subset \partial U$$

Observación. La definición 2.18 es aplicable solamente a la ecuación de difusión. No aplica a la ecuación de transporte.

Definimos el conjunto $\Gamma_V \subset \partial U$ como el lugar geométrico de todos los puntos $\mathbf{x} \in \partial U$ donde imponemos esta condición de contorno. Esta condición es de tipo Robin ya que se da el valor de una combinación lineal de la incógnita ϕ y de su derivada normal $\partial\phi/\partial n$.

Definición 2.19. En forma análoga a la definición 2.15, llamamos *condición de contorno de reflexión o de simetría* cuando la corriente neta entrante en el punto $\mathbf{x} \in \partial U$ es igual a cero. En este caso, por la ley de Fick debe anularse la derivada normal del flujo escalar evaluada en $\mathbf{x}$:

$$\operatorname{grad}\big[\phi(\mathbf{x}, E, t)\big] \cdot \hat{\mathbf{n}} = \frac{\partial \phi}{\partial n} = 0 \qquad \forall \mathbf{x} \in \Gamma_M \subset \partial U$$

Esta es una condición de tipo Neumann homogénea. Definimos el conjunto $\Gamma_M \subset U$ como el lugar geométrico de todos los puntos $\mathbf{x} \in \partial U$ donde imponemos esta condición de contorno.

Para el problema de difusión, a veces se suele utilizar una tercera condición de contorno basada en la idea que sigue. Si extrapolamos linealmente el flujo escalar ϕ una distancia ζ (que depende de la posición $\mathbf{x}$ y de la energía E) en la dirección de la normal externa $\hat{\mathbf{n}}$ en la frontera ∂U mediante una expansión de Taylor a primer orden, tenemos

$$\phi(\mathbf{x} + \zeta(\mathbf{x}, E) \cdot \hat{\mathbf{n}}, E, t)\Big|_{\mathbf{x} \in \partial U} \approx \phi(\mathbf{x}, E, t) + \frac{\partial \phi}{\partial n} \cdot \zeta(\mathbf{x}, E)$$

Si se cumple la condición de contorno de vacío dada por la definición 2.18

$$\phi(\mathbf{x}, E, t) + 2 \cdot D(\mathbf{x}, E) \cdot \frac{\partial \phi}{\partial n} = 0$$

entonces el flujo escalar extrapolado se anula en el punto

$$\mathbf{x}^* = \mathbf{x} + 2 \cdot D(\mathbf{x}, E) \cdot \hat{\mathbf{n}}$$

como ilustramos en la figura 2.10.

Si el valor numérico del coeficiente de difusión $D(\mathbf{x}, E)$ (que tiene unidades de longitud) es mucho menor que el tamaño característico del dominio U entonces podemos aproximar $\zeta \approx 0$.

Definición 2.20. Llamamos *condición de contorno de flujo nulo* cuando el flujo escalar ϕ se anula un punto $\mathbf{x} \in \partial U$:

$$\phi(\mathbf{x}, E, t) = 0 \qquad \forall \mathbf{x} \in \Gamma_N \subset \partial U$$

Definimos el conjunto $\Gamma_N \subset \partial U$ como el lugar geométrico de todos los puntos $\mathbf{x} \in \partial U$ donde imponemos esta condición de contorno. Matemáticamente esta condición es de tipo Dirichlet homogénea.

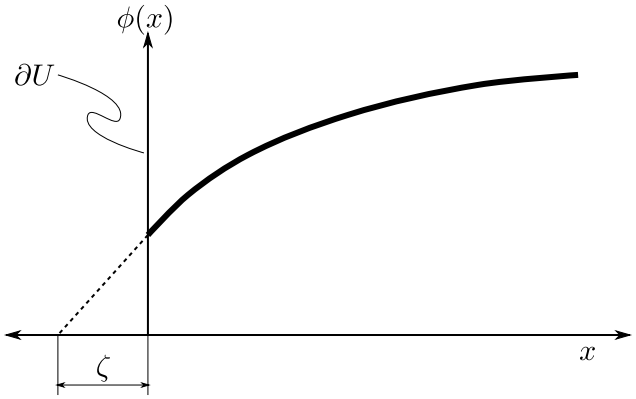


Figura 2.10.: El flujo escalar $\phi(\mathbf{x}, E, t)$ extrapolado linealmente se anula a una distancia $\zeta(\mathbf{x}, E) = 2D(\mathbf{x}, E)$ en una condición de contorno tipo vacío de la ecuación de difusión.

2.5. Esquema de solución multi-escala

La ecuación de transporte ecuación 2.29 describe completamente la interacción de neutrones con la materia y es exacta mientras

a. la población neutrónica sea lo suficientemente grande como para que podamos asumir que el flujo angular ψ es determinista, y
b. se cumplan las siete suposiciones listadas al comienzo del capítulo(página 30).

Los coeficientes tanto de la ecuación de transporte como de la ecuación de difusión son las secciones eficaces macroscópicas de los materiales presentes en el dominio U, que en principio son el producto de la sección eficaz microscópica por la densidad volumétrica de cada uno de los isótopos que componen dichos materiales.

Un neutrón nacido por fisión tiene una energía de aproximadamente 2 MeV, y cuando llega al equilibrio térmico con el medio puede alcanzar una energía de 0.02 eV. Esto es, la variable independiente E usualmente abarca ocho órdenes de magnitud. Recordando la figura 2.3, las secciones eficaces microscópicas pueden cambiar cinco o incluso seis órdenes de magnitud en este rango de energías, abarcando resonancias extremadamente difíciles de modelar matemáticamente.

Estas variaciones hace que sea prácticamente imposible resolver directamente la ecuación de transporte para obtener la dependencia del flujo angular ψ con el espacio, la dirección, la energía y eventualmente el tiempo sobre el dominio U a partir de las secciones eficaces microscópicas y de las concentraciones volumétricas de los isótopos. En la práctica se recurre a un esquema multi-escala, donde primero se evalúan y procesan las secciones eficaces microscópicas experimentales. Luego se evalúan celdas típicas de los componentes de los reactores nucleares (elementos combustibles, barras de control, etc.) para obtener secciones eficaces condensadas que finalmente son los coeficientes de la ecuación de transporte (o difusión) utilizada para realizar un cálculo a nivel de núcleo.[12]

[12]En el sentido del inglés *core*.

2.5.1. Evaluación y procesamiento de secciones eficaces

Este nivel de cálculo es el punto central del trabajo de doctorado de la referencia [37]. A partir de mediciones experimentales, los datos se procesan de forma tal de que puedan ser evaluados y utilizados como bibliotecas de secciones eficaces microscópicas con las cuales realizar cálculos a nivel de celda.

2.5.2. Cálculo a nivel celda

Este nivel de cálculo es el punto central del trabajo de doctorado de la referencia [33]. A partir de bibliotecas de secciones eficaces microscópicas se procede a realizar cálculos de transporte de forma tal de calcular secciones eficaces macroscópicas a pocos grupos de energía que puedan ser usadas como los coeficientes de las ecuaciones de transporte utilizadas en el cálculo a nivel de núcleo.

2.5.3. Cálculo a nivel núcleo

Este nivel de cálculo es el punto central de esta tesis, especialmente en los capítulos 3 y 5. Las secciones eficaces macroscópicas que son los coeficientes de las ecuaciones de ordenadas discretas y difusión de neutrones multigrupo se suponen funciones conocidas del espacio.

3. Esquemas de discretización numérica

> You can know a great deal about something without writing about it. Can you ever know so much that you wouldn't learn more from trying to explain what you know? I don't think so. I've written about at least two subjects I know well—Lisp hacking and startups—and in both cases I learned a lot from writing about them.[a] In both cases there were things I didn't consciously realize till I had to explain them. And I don't think my experience was anomalous. A great deal of knowledge is unconscious, and experts have if anything a higher proportion of unconscious knowledge than beginners.
> *Paul Graham, Putting Ideas into Words, 2022*
>
> [a] Machinery and circuits are formal languages.

> Boundary conditions tend to make the theory of PDEs difficult.
> *Jürgen Jost, Partial Differential Equations, 2013* [26]

En el capitulo anterior hemos arribado a formulaciones matemáticas que modelan los procesos físicos de transporte y difusión de neutrones en estado estacionario mediante ecuaciones integro-diferenciales. Bajo las suposiciones que explicitamos al comienzo del capítulo 2 y asumiendo que las secciones eficaces macroscópicas son funciones del espacio y de la energía conocidas, estas ecuaciones son *exactas*. Para la ecuación de difusión, que es de segundo orden pero más sencilla de resolver, llegamos a

$$\begin{gathered} -\mathrm{div}\Big[D(\mathbf{x},E)\cdot \mathrm{grad}\left[\phi(\mathbf{x},E)\right]\Big] + \Sigma_t(\mathbf{x},E)\cdot\phi(\mathbf{x},E) = \\ \int_0^\infty \Sigma_{s_0}(\mathbf{x},E'\to E)\cdot\phi(\mathbf{x},E')\,dE' + \chi(E)\int_0^\infty \nu\Sigma_f(\mathbf{x},E')\cdot\phi(\mathbf{x},E')\,dE' + s_0(\mathbf{x},E) \end{gathered} \tag{2.62}$$

y para la ligeramente más compleja ecuación de transporte linealmente anisótropa obtuvimos

$$\begin{aligned}
&\hat{\mathbf{\Omega}} \cdot \operatorname{grad}\left[\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E)\right] + \Sigma_t(\mathbf{x}, E) \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E) = \\
&\frac{1}{4\pi} \cdot \int_0^\infty \Sigma_{s_0}(\mathbf{x}, E' \to E) \cdot \int_{4\pi} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E')\, d\hat{\mathbf{\Omega}}'\, dE' + \\
&\frac{3 \cdot \hat{\mathbf{\Omega}}}{4\pi} \cdot \int_0^\infty \Sigma_{s_1}(\mathbf{x}, E' \to E) \cdot \int_{4\pi} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E') \cdot \hat{\mathbf{\Omega}}'\, d\hat{\mathbf{\Omega}}'\, dE' \\
&+ \frac{\chi(E)}{4\pi} \int_0^\infty \nu\Sigma_f(\mathbf{x}, E') \cdot \int_{4\pi} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}', E')\, d\hat{\mathbf{\Omega}}'\, dE' + s(\mathbf{x}, \hat{\mathbf{\Omega}}, E)
\end{aligned} \tag{2.41}$$

sobre un espacio de fases generado[1] por seis escalares independientes:

- tres para el espacio $\mathbf{x}$,
- dos para la dirección $\hat{\mathbf{\Omega}}$ y
- uno para la energía E.

El objetivo de este capítulo es transformar estas dos ecuaciones diferenciales en derivadas parciales en sistemas de ecuaciones algebraicas de tamaño finito de forma tal que las podamos resolver con una herramienta computacional, cuya implementación describimos en el capítulo 4. Este proceso involucra inherentemente aproximaciones relacionadas a la discretización de la energía E, la dirección $\hat{\mathbf{\Omega}}$ y el espacio $\mathbf{x}$, por lo que las soluciones a las ecuaciones diferenciales que podamos encontrar numéricamente serán solamente aproximaciones a las soluciones matemáticas reales. Según discutimos en la sección 3.1, estas aproximaciones serán mejores a medida que aumentemos la cantidad de entidades discretas. Pero al mismo tiempo aumentan los recursos y costos de ingeniería asociados.

Comenzamos primero entonces introduciendo algunas propiedades matemáticas de los métodos numéricos y discutiendo cuestiones a tener en cuenta para analizarlos desde el punto de vista del gerenciamiento de proyectos de ingeniería. Pasamos luego a la discretización de las ecuaciones propiamente dicha. Primeramente discretizamos la dependencia en energía aplicando la idea de grupos discretos de energías para obtener las llamadas "ecuaciones multigrupo". Continuamos luego por la dependencia angular de la ecuación de transporte aplicando el método de ordenadas discretas S_N. Esencialmente la idea es transformar las integrales sobre E' y sobre $\hat{\mathbf{\Omega}}'$ en las dos ecuaciones 2.62 y 2.41 del principio del capítulo por sumatorias finitas.

El grueso del capítulo lo dedicamos a la discretización espacial de ambas ecuaciones, que es el aporte principal de esta tesis al problema de la resolución de las ecuaciones de transporte de neutrones a nivel de núcleo utilizando mallas no estructuradas y técnicas de descomposición de dominio para permitir la resolución de problemas de tamaño arbitrario. En la monografía [62] mostramos, para la ecuación de difusión, una derivación similar a la formulación propuesta en esta tesis basada en elementos finitos. Pero también se incluye una formulación espacial basada en volúmenes finitos. Por cuestiones de longitud, hemos decidido enfocarnos solamente en elementos finitos en esta tesis. Dejamos la extensión a volúmenes finitos y su comparación con otros esquemas como trabajos futuros.

Finalmente analizamos la forma matricial/vectorial de los tres casos de problemas de estado estacionario que resolvemos en esta tesis:

[1] Del inglés *spanned*.

- medio no multiplicativo con fuentes independentes
- medio multiplicativo con fuentes independientes
- medio multiplicativo sin fuentes independientes

3.1. Métodos numéricos

En forma general, las ecuaciones 2.62 y 2.41 que derivamos en el capítulo anterior a partir de primeros principios están expresadas en una formulación fuerte (ver definición 3.12) y exacta

$$\mathcal{F}(\varphi, \Sigma) = 0$$

denotando con

- φ el flujo incógnita (ψ o ϕ) "exacto"[2] que depende continuamente de $\mathbf{x}$, E y $\hat{\mathbf{\Omega}}$,
- Σ todos los datos de entrada, incluyendo el dominio espacial de dimensión D continuo $U \in \mathbb{R}^D$ y las secciones eficaces con sus dependencias continuas de $\mathbf{x}$, E y $\hat{\mathbf{\Omega}}$,
- $\mathcal{F}$ un operador integral sobre E' y $\hat{\mathbf{\Omega}}'$ y diferencial sobre $\mathbf{x}$

Esencialmente, en este capítulo aplicamos métodos numéricos [44] para obtener una formulación débil (ver definición 3.13) y aproximada

$$\mathcal{F}_N(\varphi_N, \Sigma_N) = 0 \tag{3.1}$$

donde ahora

- φ_N es una aproximación discreta de tamaño N del flujo incógnita,
- Σ_N es una aproximación de los datos de entrada, incluyendo una discretización U_N del dominio espacial
- $\mathcal{F}_N$ es un operador discreto de tamaño N

El tamaño N del operador discreto $\mathcal{F}_N$ es el producto de

a. la cantidad G de grupos de energías (sección 3.2),
b. la cantidad M de direcciones de vuelo discretas (sección 3.3), y
c. la cantidad J de incógnitas espaciales (sección 3.4).

Definición 3.1 (Convergencia). Un método numérico es *convergente* si

$$\lim_{N\to\infty} ||\varphi - \varphi_N|| = 0$$

para alguna norma apropiada $||\cdot||$. Por ejemplo, para la norma L_2:

$$\lim_{N\to\infty} \sqrt{\int_U \int_{4\pi} \int_0^\infty \left[\varphi(\mathbf{x}, \hat{\mathbf{\Omega}}, E) - \varphi_N(\mathbf{x}, \hat{\mathbf{\Omega}}, E)\right]^2 dE\, d\hat{\mathbf{\Omega}}\, d^3\mathbf{x}} = 0$$

[2] En el sentido matemático de satisfacer exactamente la ecuación diferencial. El análisis de la exactitud física queda fuera del alcance de esta tesis.

La convergencia y, más aún, el orden con el cual el error $||\varphi - \varphi_N||$ converge a cero es importante al verificar la implementación computacional de un método numérico. Tanto es así que para que una herramienta computacional sea verificada en el sentido de "verificación y validación" de software, no sólo se tiene que mostrar que $\lim_{N\to\infty} ||\varphi - \varphi_N|| = 0$ sino que la tasa de disminución de este error con $1/N$ tiene que coincidir con el orden del método numérico (ver sección 5.9). De todas maneras, demostrar que un método numérico genérico es convergente no es sencillo y ni siquiera posible en la mayoría de los casos. En forma equivalente, se prueban los conceptos de consistencia y estabilidad definidos a continuación y luego se utiliza el teorema de equivalencia que sigue.

Definición 3.2 (Consistencia). Un método numérico es *consistente* si

$$\lim_{N\to\infty} \mathcal{F}_N(\varphi, \Sigma) = \lim_{N\to\infty} [\mathcal{F}_N(\varphi, \Sigma) - \mathcal{F}(\varphi, \Sigma)] = 0$$

dado que $\mathcal{F}(\varphi, \Sigma) = 0$. Es decir, si el operador discreto $\mathcal{F}_N$ tiende al operador continuo $\mathcal{F}$ para $N \to \infty$ entonces el método es consistente. Más aún, si

$$\mathcal{F}_N(\varphi, \Sigma) = [\mathcal{F}_N(\varphi, \Sigma) - \mathcal{F}(\varphi, \Sigma)] = 0 \quad \forall N \geq 1$$

entonces decimos que el método numérico es *fuertemente*[3] o *completamente*[4] consistente.

Definición 3.3 (Estabilidad). Un método numérico es *estable* si dada una perturbación pequeña $\delta\Sigma_N$ en los datos de entrada tal que

$$\mathcal{F}_N(\varphi_N + \delta\varphi_N, \Sigma_N + \delta\Sigma_N) = 0$$

entonces la perturbación $\delta\varphi_N$ causada en la solución también es pequeña. Formalmente, un método numérico es estable si

$$\forall \epsilon > 0, \exists \delta(\epsilon) > 0 : \forall \delta\Sigma_N \;/\; ||\delta\Sigma_N|| < \delta(\epsilon) \Rightarrow ||\delta\varphi_N|| < \epsilon \quad \forall N \geq 1$$

La consistencia es relativamente sencilla de demostrar. La estabilidad es un poco más compleja, pero posible. Finalmente, la convergencia queda demostrada a partir del siguiente resultado.

Teorema 3.1 (de equivalencia de Lax-Richtmyer). *Si un método numérico es consistente, entonces es convergente si y sólo si es estable. Más aún, cualesquiera dos propiedades implica la tercera.*

[3] Del inglés *strongly*.
[4] Del inglés *fully*.

3.1.1. Comparaciones y evaluaciones económicas

Suponiendo que disponemos de varios métodos numéricos que nos permitan calcular φ_N a partir de un conjunto de datos de entrada Σ_N sobre un cierto espacio de fases discretizado, cabría preguntarnos cuál es el más eficiente para resolver un cierto problema de ingeniería nuclear. Está claro que en este sentido, la eficiencia depende de

1. la exactitud de la solución φ_N obtenida
2. los recursos computacionales necesarios para obtener φ_N, medidos en
    a. tiempo total de procesamiento (CPU, GPU y/o APU)
    b. tiempo de pared,[5] que es igual al del punto a en serie pero debería ser menor en cálculos en paralelo,
    c. memoria RAM,
    d. necesidades de almacenamiento, etc.
3. los recursos humanos necesarios para
    a. preparar Σ_N (pre-procesar),
    b. analizar φ_N (post-procesar), y
    c. llegar a conclusiones útiles.

Si bien con esta taxonomía pareciera ser que comparar métodos numéricos no debería ser muy difícil, hay detalles que deben ser tenidos en cuenta y que de hecho complican la evaluación. Por ejemplo, dado un cierto problema de análisis de reactores a nivel de núcleo, el punto 1 incluye las siguiente preguntas:

- ¿Es necesario resolver la ecuación de transporte o la ecuación de difusión es suficiente?
- ¿Cuántas direcciones discretas hay que tener en cuenta para obtener una exactitud apropiada?

Por otro lado, el punto 2 abarca cuestiones como

- ¿Es más eficiente discretizar el espacio con una formulación precisa como Galerkin que da lugar a matrices no simétricas usando pocos grados de libertad o conviene utilizar una formulación menos precisa como cuadrados mínimos que da lugar a matrices simétricas pero empleando más incógnitas espaciales?
- ¿Es preferible usar métodos directos que son robustos pero poco escalables o métodos iterativos que son escalables pero sensibles a perturbaciones?

La determinación del valor de N necesario para contar con una cierta exactitud apropiada para cada método numérico no es trivial e involucra estudios paramétricos para obtener φ_N vs. N. Este proceso puede necesitar barrer valores de N suficientemente grandes para los cuales haya discontinuidades en la evaluación. Por ejemplo, si se debe pasar de una sola computadora a más de una por limitaciones de recursos (usualmente memoria RAM) o si se debe pasar de una infra-estructura *on-premise* a una basada en la nube en un eventual caso donde se necesiten más nodos (*hosts*) de cálculo que los disponibles.

[5] En el sentido del inglés *wall time*.

Finalmente, como en cualquier evaluación técnico-económica, intervienen situaciones particulares más blandas relacionadas al gerenciamiento de proyectos y a la tensión de los tres vértices del triángulo alcance-costos-tiempo, como por ejemplo:

- ¿Se necesitan resultados precisos (caros y lentos) o resultados aproximados (baratos y rápidos) son suficientes?
- ¿Se prioriza disminuir los costos (como en la mayoría de los proyectos de ingeniería) o se prioriza tener resultados en poco tiempo (e.g. Proyecto Manhattan [49])?
- ¿Cómo dependen los tiempos y los costos de la infra-estructura de los recursos computacionales?
  - Si es *on-premise*:
    - amortización de hardware
    - mantenimiento de hardware
    - licencias de software
    - administración de software
    - energía eléctrica
  - Si es *cloud*:
    - alquiler de instancias
    - suscripciones a servicios
    - orquestación
- ¿Cómo son los costos asociados a la capacitación de los ingenieros que tienen que obtener φ con cada método numérico?

Está claro que el análisis de todas estas combinaciones están fuera del alcance de esta tesis. De todas maneras, la herramienta computacional cuya implementación describimos en detalle en el capítulo 4 permite evaluar todos estos aspectos y muchos otros ya que, en forma resumida

1. Está diseñado para ser ejecutado nativamente en la nube.[6]
2. Permite discretizar el dominio espacial utilizando mallas no estructuradas.[7]
3. Puede correr en paralelo en una cantidad arbitraria de computadoras.[8]

En particular, permite a los ingenieros nucleares comparar las soluciones obtenidas con las formulaciones S_N y de difusión al resolver un mismo problema de tamaño arbitrario. De esta forma, es posible justificar ante gerencias superiores o entes regulatorios la factibilidad (o no) de encarar un proyecto para analizar un reactor nuclear con la ecuación de difusión utilizando

a. datos objetivos, y
b. juicio de ingeniería.

[6] Del inglés *cloud native* como contrapartida a *cloud friendly* o *cloud enabled*.
[7] Del inglés *unstructured grids*.
[8] Del inglés *hosts*.

3.2. Discretización en energía

Vamos a discretizar el dominio de la energía $E \in \mathbb{R}$ utilizando el concepto clásico de física de reactores de *grupos de energías*, que llevado a conceptos más generales de matemática discreta es equivalente a integrar sobre volúmenes (intervalos en $\mathbb{R}$) de control y utilizar el valor medio sobre cada volumen como el valor discretizado.

En efecto, tomemos el intervalo de energías $[0, E_0]$ donde E_0 es la mayor energía esperada de un neutrón individual. Como ilustramos en la figura 3.1, dividamos dicho intervalo en G grupos (volúmenes) no necesariamente iguales, cada uno definido por energías de corte $0 = E_G < E_{G-1} < \cdots < E_2 < E_1 < E_0$, de forma tal que el grupo g es el intervalo $[E_g, E_{g-1}]$.

Observación. Con esta notación, el grupo número uno siempre es el de mayor energía. A medida que un neutrón va perdiendo energía, va aumentando el número de su grupo de energía.

Figura 3.1.: Discretización del dominio energético en grupos (volúmenes) de energía. Tomamos la mayor energía esperada E_0 y dividimos el intervalo $[0, E_0]$ en G grupos, no necesariamente iguales. El grupo uno es el de mayor energía.

Definición 3.4. El flujo angular ψ_g del grupo g es

$$\psi_g(\mathbf{x}, \hat{\mathbf{\Omega}}) = \int_{E_g}^{E_{g-1}} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E)\, dE$$

Definición 3.5. El flujo escalar ϕ_g del grupo g es

$$\phi_g(\mathbf{x}) = \int_{E_g}^{E_{g-1}} \phi(\mathbf{x}, E)\, dE$$

Definición 3.6. El vector corriente $\mathbf{J}_g$ del grupo g es

$$\mathbf{J}_g(\mathbf{x}) = \int_{E_g}^{E_{g-1}} \mathbf{J}(\mathbf{x}, E)\, dE = \int_{E_g}^{E_{g-1}} \int_{4\pi} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E) \cdot \hat{\mathbf{\Omega}}\, d\hat{\mathbf{\Omega}}\, dE = \int_{4\pi} \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}}) \cdot \hat{\mathbf{\Omega}}\, d\hat{\mathbf{\Omega}} =$$

Observación. Los flujos $\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E)$ y $\psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})$ no tienen las mismas unidades. La primera magnitud tiene unidades de inversa de área por inversa de ángulo sólido por inversa de energía por inversa de tiempo (por ejemplo $\mathrm{cm}^{-2} \cdot \mathrm{eV}^{-1} \cdot \mathrm{s}^{-1}$), mientras que la segunda es un flujo integrado por lo que sus unidades son inversa de área por inversa de ángulo sólido por inversa de tiempo (por ejemplo $\mathrm{cm}^{-2} \cdot \mathrm{s}^{-1}$). La misma idea aplica a $\phi(\mathbf{x}, E)$ y a $\phi_g(\mathbf{x})$.

Los tres objetivos de discretizar la energía en G grupos son

1. transformar la dependencia continua del flujo angular $\psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E)$ con la energía E en G funciones $\psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})$ y del flujo escalar $\phi(\mathbf{x}, E)$ en G funciones $\phi_g(\mathbf{x})$,
2. reemplazar las integrales sobre la variable continua E' por sumatorias finitas sobre el índice g', y
3. re-escribir las ecuaciones de difusión y transporte en función de los flujos de grupo ($\psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})$ en transporte y $\phi_g(\mathbf{x})$ en difusión)

Para ilustrar la idea, prestemos atención al término de absorción total de la ecuación de transporte $\Sigma_t \cdot \psi$. El objetivo es integrarlo con respecto a E entre E_g y E_{g-1} y escribirlo como el producto de una sección eficaz total asociada al grupo g por el flujo angular ψ_g de la definición 3.4:

$$\int_{E_g}^{E_{g-1}} \Sigma_t(\mathbf{x}, E) \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E)\, dE = \Sigma_{tg}(\mathbf{x}) \cdot \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}}) \tag{3.2}$$

De la misma manera, para la ecuación de difusión quisiéramos que

$$\int_{E_g}^{E_{g-1}} \Sigma_t(\mathbf{x}, E) \cdot \phi(\mathbf{x}, E)\, dE = \Sigma_{tg}(\mathbf{x}) \cdot \phi_g(\mathbf{x})$$

Según la definición 3.4, la sección eficaz total Σ_{tg} media en el grupo g debe ser

$$\Sigma_{tg}(\mathbf{x}) = \frac{\displaystyle\int_{E_g}^{E_{g-1}} \Sigma_t(\mathbf{x}, E) \cdot \phi(\mathbf{x}, E)\, dE}{\displaystyle\int_{E_g}^{E_{g-1}} \phi(\mathbf{x}, E)\, dE}$$

con lo que no hemos ganado nada ya que llegamos a una condición tautológica donde el parámetro que necesitamos para no tener que conocer la dependencia explícita del flujo con la energía depende justamente de dicha dependencia. Sin embargo —y es ésta una de las ideas centrales del cálculo y análisis de reactores— podemos suponer que el cálculo de celda (sección 2.5.2) es capaz de proveernos las secciones eficaz macroscópicas multigrupo para el reactor que estamos modelando de forma tal que, desde el punto de vista del cálculo de núcleo, Σ_{tg} y todas las demás secciones eficaces macroscópicas son distribuciones conocidas del espacio $\mathbf{x}$.

Para analizar la sección eficaz de ν-fisiones, integremos el término de fisión de la ecuación de transporte entre las energías E_{g-1} y E_g e igualémoslo a una sumatoria de productos $\nu\Sigma_{fg'} \cdot \phi_{g'}$[9]

$$\int_{E_{g-1}}^{E_g} \frac{\chi(E)}{4\pi} \cdot \int_0^{\infty} \nu\Sigma_f(\mathbf{x}, E') \cdot \phi(\mathbf{x}, E')\, dE'\, dE = \frac{\chi_g}{4\pi} \cdot \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \cdot \phi_{g'}(\mathbf{x}) \tag{3.3}$$

[9] Podríamos haber integrado la ecuación de difusión, en cuyo caso no tendríamos el denominador 4π en ambos miembros. En cualquier caso, el resultado sería el mismo.

entonces

$$\chi_g = \int_{E_{g-1}}^{E_g} \chi(E)\, dE \tag{3.4}$$

y

$$\nu\Sigma_{fg}(\mathbf{x}) = \frac{\displaystyle\int_{E'_g}^{E'_{g-1}} \nu\Sigma_f(\mathbf{x}, E') \cdot \phi(\mathbf{x}, E')\, dE'}{\displaystyle\int_{E'_g}^{E'_{g-1}} \phi(\mathbf{x}, E')\, dE'}$$

Para el término de scattering isotrópico, requerimos que

$$\int_{E_{g-1}}^{E_g} \frac{1}{4\pi} \cdot \int_0^{\infty} \Sigma_{s_0}(\mathbf{x}, E' \to E) \cdot \phi(\mathbf{x}, E')\, dE'\, dE = \frac{1}{4\pi} \cdot \sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \cdot \phi_{g'}(\mathbf{x}) \tag{3.5}$$

entonces

$$\Sigma_{s_0 g' \to g}(\mathbf{x}) = \frac{\displaystyle\int_{E_{g-1}}^{E_g} \int_{E'_{g-1}}^{E'_g} \Sigma_{s_0}(\mathbf{x}, E' \to E) \cdot \phi(\mathbf{x}, E')\, dE}{\displaystyle\int_{E'_{g-1}}^{E'_g} \phi(\mathbf{x}, E')\, dE'}$$

Observación. Necesitamos una doble integral sobre E y sobre E' porque $\Sigma_{s_0}(\mathbf{x}, E' \to E)$ es una sección eficaz diferencial y tiene unidades de inversa de longitud por inversa de ángulo sólido por inversa de energía.

Un análisis similar para el término de scattering linealmente anisótropo

$$\int_{E_{g-1}}^{E_g} \frac{3 \cdot \hat{\mathbf{\Omega}}}{4\pi} \cdot \int_0^{\infty} \Sigma_{s_0}(\mathbf{x}, E' \to E) \cdot \mathbf{J}(\mathbf{x}, E')\, dE'\, dE = \frac{3 \cdot \hat{\mathbf{\Omega}}}{4\pi} \cdot \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \cdot \mathbf{J}_{g'}(\mathbf{x}) \tag{3.6}$$

arrojaría la necesidad de pesar la sección eficaz diferencial con la corriente $\mathbf{J}$ en lugar de con el flujo escalar ϕ, dejando una expresión sin sentido matemático como

$$\Sigma_{s_1 g' \to g}(\mathbf{x}) = \frac{\displaystyle\int_{E_{g-1}}^{E_g} \int_{E'_{g-1}}^{E'_g} \Sigma_{s_1}(\mathbf{x}, E' \to E) \cdot \mathbf{J}(\mathbf{x}, E')\, dE}{\displaystyle\int_{E'_{g-1}}^{E'_g} \mathbf{J}(\mathbf{x}, E')\, dE'}$$

a menos que tanto numerador como denominador tengan sus elementos proporcionales entre sí y la división se tome como elemento a elemento. Usualmente se desprecia la diferencia entre corriente y flujo y podemos utilizar el flujo para pesar el término de scattering anisótropo:

$$\Sigma_{s_1 g' \to g}(\mathbf{x}) \approx \frac{\displaystyle\int_{E_{g-1}}^{E_g} \int_{E'_{g-1}}^{E'_g} \Sigma_{s_1}(\mathbf{x}, E' \to E) \cdot \phi(\mathbf{x}, E')\, dE}{\displaystyle\int_{E'_{g-1}}^{E'_g} \phi(\mathbf{x}, E')\, dE'}$$

Integremos ahora la ecuación de transporte ecuación 2.41 con respecto a E entre E_g y E_{g-1}:

$$\begin{aligned}
\hat{\mathbf{\Omega}} \cdot \operatorname{grad} \left[\int_{E_g}^{E_{g-1}} \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E)\, dE \right] + \int_{E_g}^{E_{g-1}} \Sigma_t(\mathbf{x}, E) \cdot \psi(\mathbf{x}, \hat{\mathbf{\Omega}}, E)\, dE = \\
\int_{E_g}^{E_{g-1}} \frac{1}{4\pi} \cdot \int_0^{\infty} \Sigma_{s_0}(\mathbf{x}, E' \to E) \cdot \phi(\mathbf{x}, E')\, dE' + \\
\int_{E_g}^{E_{g-1}} \frac{3 \cdot \hat{\mathbf{\Omega}}}{4\pi} \cdot \int_0^{\infty} \Sigma_{s_1}(\mathbf{x}, E' \to E) \cdot \mathbf{J}(\mathbf{x}, E')\, dE' + \\
\int_{E_g}^{E_{g-1}} \frac{\chi(E)}{4\pi} \int_0^{\infty} \int_{4\pi} \nu\Sigma_f(\mathbf{x}, E') \cdot \phi(\mathbf{x}, E')\, dE'\, dE + \int_{E_g}^{E_{g-1}} s(\mathbf{x}, \hat{\mathbf{\Omega}}, E)\, dE
\end{aligned}$$

Definición 3.7. Definimos la fuente de neutrones independientes del grupo g como

$$s_g(\mathbf{x}, \hat{\mathbf{\Omega}}) = \int_{E_g}^{E_{g-1}} s(\mathbf{x}, \hat{\mathbf{\Omega}}, E)\, dE$$

Definición 3.8. Definimos el momento de orden cero de las fuentes independientes del grupo g como

$$s_{0g}(\mathbf{x}) = \int_{E_g}^{E_{g-1}} s_0(\mathbf{x}, E)\, dE$$

Teniendo en cuenta las definiciones

- 3.4 (flujo angular del grupo g)
- 3.5 (flujo escalar del grupo g)
- 3.6 (corriente del grupo g)
- 3.7 (fuente del grupo g)

y las ecuaciones

- 3.2 (ritmo de absorciones)
- 3.3 (ritmo de fisiones)

- 3.4 (espectro de fisiones)
- 3.5 (scattering isotrópico)
- 3.6 (scattering linealmente anisótropo)

obtenemos las G ecuaciones de transporte multigrupo

$$\begin{aligned}\hat{\mathbf{\Omega}} \cdot \text{grad}\left[\psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\right] + \Sigma_{tg}(\mathbf{x}) \cdot \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}}) = \frac{1}{4\pi} \cdot \sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \cdot \phi_{g'}(\mathbf{x}) + \\ \frac{3 \cdot \hat{\mathbf{\Omega}}}{4\pi} \cdot \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \cdot \mathbf{J}_{g'}(\mathbf{x}) + \frac{\chi_g}{4\pi} \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \cdot \phi_{g'}(\mathbf{x}) + s_g(\mathbf{x}, \hat{\mathbf{\Omega}})\end{aligned} \tag{3.7}$$

donde las incógnitas son $\psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})$ para $g = 1, \dots, G$

Procediendo de forma análoga para la ecuación de difusión ecuación 2.62, primero integrándola con respecto a E entre E_{g-1} y E_g y luego teniendo en cuenta las definiciones

- 3.4 (flujo angular del grupo g)
- 3.5 (flujo escalar del grupo g)
- 3.7 (fuente del grupo g)

podemos obtener la ecuación de difusión multigrupo

$$\begin{aligned}-\text{div}\Big[D_g(\mathbf{x}) \cdot \text{grad}\left[\phi_g(\mathbf{x})\right]\Big] + \Sigma_{tg}(\mathbf{x}) \cdot \phi_g(\mathbf{x}) = \\ \sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \cdot \phi_{g'}(\mathbf{x}) + \chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \cdot \phi_{g'}(\mathbf{x}) + s_{0g}(\mathbf{x})\end{aligned} \tag{3.8}$$

donde ahora las incógnitas son $\phi_g(\mathbf{x})$ para $g = 1, \dots, G$,

Observación. El coeficiente de difusión D_g del grupo g proviene de calcular las secciones eficaces Σ_{tg}, Σ_{st} y el coseno medio de scattering μ_{0g} del grupo g y reemplazar la definición 2.17 por

$$D_g(\mathbf{x}) = \frac{1}{3\left[\Sigma_{tg}(\mathbf{x}) - \mu_{0g}(\mathbf{x}) \cdot \Sigma_{s_t g}(\mathbf{x})\right]}$$

Observación. Matemáticamente, la aproximación multigrupo es equivalente a discretizar el dominio de la energía con un esquema de volúmenes finitos con la salvedad de que no hay operadores diferenciales con respecto a la variable E sino que el acople entre volúmenes se realiza en forma algebraica. Dicho acople no es necesariamente entre primeros vecinos solamente sino que es arbitrario, i.e. un neutrón puede pasar del grupo 1 al G, o viceversa, o de un grupo arbitrario g' a otro grupo g.

Observación. Dado que en las ecuaciones multigrupo 3.7 y 3.8 la discretización es estrictamente algebraica y deliberadamente tautológica, la consistencia es teóricamente fuerte ya que el operador discretizado coincide con el operador continuo incluso para un único grupo de energías $G = 1$. De hecho las ecuaciones multigrupo se basan solamente en *definiciones*. En la práctica, la consistencia depende del cálculo a nivel de celda de la sección 2.5.2.

3.3. Discretización en ángulo

Para discretizar la dependencia espacial de la ecuación de transporte multigrupo 3.7 aplicamos el método de ordenadas discretas o S_N, discutido en la literatura tradicional de física de reactores. En esta tesis lo derivamos al integrar las ecuaciones multigrupo continuas en $\hat{\mathbf{\Omega}}$ sobre volúmenes de control finitos como si fuese un esquema numérico basado en el método de volúmenes finitos. De hecho, en este caso, los volúmenes finitos son áreas $\Delta\hat{\mathbf{\Omega}}_m$ discretas de la esfera unitaria donde cada una de ellas tiene asociadas

1. un peso w_m
2. una dirección particular $\hat{\mathbf{\Omega}}_m$, y
3. una fracción de total de área unitaria $\Delta\hat{\mathbf{\Omega}}_m/4\pi$

para $m = 1, \dots, M$. Nuevamente el acople entre volúmenes de control es algebraico y no necesariamente a primeros vecinos.

Observación. La cantidad $\Delta\hat{\mathbf{\Omega}}_m$ es un escalar ya que representa una porción de área de la esfera unitaria alrededor del versor $\hat{\mathbf{\Omega}}_m$.

Teorema 3.2 (de cuadratura sobre la esfera unitaria). *La integral de una función escalar $f(\hat{\mathbf{\Omega}})$ de cuadrado integrable sobre todas las direcciones $\hat{\mathbf{\Omega}}$ es igual a 4π veces la suma de un conjunto de M pesos w_m normalizados tal que $\sum w_m = 1$, multiplicados por M valores medios $\left\langle f(\hat{\mathbf{\Omega}})\right\rangle_m$ asociados a M direcciones $\hat{\mathbf{\Omega}}_m$ donde cada una de las cuales tiene asociada también una porción $\Delta\hat{\mathbf{\Omega}}_m$ de la esfera unitaria tal que su unión es 4π y su intersección es cero:*

$$\int_{4\pi} f(\hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}} = 4\pi \cdot \sum_{w=1}^{M} w_m \cdot \left\langle f(\hat{\mathbf{\Omega}})\right\rangle_m$$

El peso w_m es

$$w_m = \frac{1}{4\pi} \cdot \int_{\Delta\hat{\mathbf{\Omega}}_m} d\hat{\mathbf{\Omega}} = \frac{\Delta\hat{\mathbf{\Omega}}_m}{4\pi}$$

Demostración. Comenzamos escribiendo la integral sobre 4π como una suma para $m = 1, \dots, M$

$$\int_{4\pi} f(\hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}} = \sum_{m=1}^{M} \int_{\Delta\hat{\mathbf{\Omega}}_m} f(\hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}}$$

Multiplicamos y dividimos por $\int_{\Delta\hat{\mathbf{\Omega}}_m} d\hat{\mathbf{\Omega}} = 4\pi \cdot w_m$

$$\sum_{m=1}^{M}\int_{\Delta\hat{\mathbf{\Omega}}_m} f(\hat{\mathbf{\Omega}})\,d\hat{\mathbf{\Omega}} = \sum_{m=1}^{M}\frac{\displaystyle\int_{\Delta\hat{\mathbf{\Omega}}_m} f(\hat{\mathbf{\Omega}})\,d\hat{\mathbf{\Omega}}}{\displaystyle\int_{\Delta\hat{\mathbf{\Omega}}_m} d\hat{\mathbf{\Omega}}}\cdot\int_{\Delta\hat{\mathbf{\Omega}}_m} d\hat{\mathbf{\Omega}} = \sum_{m=1}^{M}\frac{\displaystyle\int_{\Delta\hat{\mathbf{\Omega}}_m} f(\hat{\mathbf{\Omega}})\,d\hat{\mathbf{\Omega}}}{\displaystyle\int_{\Delta\hat{\mathbf{\Omega}}_m} d\hat{\mathbf{\Omega}}}\cdot 4\pi\, w_m$$

Si llamamos $\left\langle f(\hat{\mathbf{\Omega}})\right\rangle_{\hat{\mathbf{\Omega}}_m}$ al valor medio de f en $\Delta\hat{\mathbf{\Omega}}_m$

$$\left\langle f(\hat{\mathbf{\Omega}})\right\rangle_{\hat{\mathbf{\Omega}}_m} = \frac{\displaystyle\int_{\Delta\hat{\mathbf{\Omega}}_m} f(\hat{\mathbf{\Omega}})\,d\hat{\mathbf{\Omega}}}{\displaystyle\int_{\Delta\hat{\mathbf{\Omega}}_m} d\hat{\mathbf{\Omega}}}$$

entonces se sigue la tesis del teorema.

□

Definición 3.9. El flujo angular ψ_{mg} del grupo g asociado a la ordenada discreta m es igual al valor medio del flujo angular ψ_g del grupo g (definido en la definición 3.4) alrededor de la dirección $\hat{\mathbf{\Omega}}_m$:

$$\psi_{mg}(\mathbf{x}) = \left\langle \psi_g(\mathbf{x},\hat{\mathbf{\Omega}})\right\rangle_{\hat{\mathbf{\Omega}}_m} = \frac{\displaystyle\int_{\Delta\hat{\mathbf{\Omega}}_m} \psi_g(\mathbf{x},\hat{\mathbf{\Omega}})\,d\hat{\mathbf{\Omega}}}{\displaystyle\int_{\Delta\hat{\mathbf{\Omega}}_m} d\hat{\mathbf{\Omega}}}$$

Observación. Esta vez ψ_{mg} sí tiene la mismas unidades que ψ_g.

Corolario 3.1. *La integral del flujo escalar sobre la porción $\Delta\hat{\mathbf{\Omega}}_m$ de la esfera unitaria es 4π veces el producto $w_m \cdot \psi_{mg}$:*

$$\int_{\Delta\hat{\mathbf{\Omega}}_m} \psi_g(\mathbf{x},\hat{\mathbf{\Omega}})\,d\hat{\mathbf{\Omega}} = \psi_{mg}(\mathbf{x})\cdot\Delta\hat{\mathbf{\Omega}}_m = 4\pi\cdot w_m\cdot\psi_{mg}(\mathbf{x})$$

Corolario 3.2. *El flujo escalar ϕ_g del grupo g es igual a*

$$\phi_g(\mathbf{x}) = \int_{4\pi}\psi_g(\mathbf{x},\hat{\mathbf{\Omega}})\,d\hat{\mathbf{\Omega}} = 4\pi\sum_{m=1}^{M} w_m\cdot\psi_{mg}(\mathbf{x})$$

Re-escribamos primero la ecuación 3.7 de transporte multigrupo

$$\begin{aligned}\hat{\mathbf{\Omega}}\cdot\operatorname{grad}\left[\psi_g(\mathbf{x},\hat{\mathbf{\Omega}})\right] + \Sigma_{tg}(\mathbf{x})\cdot\psi_g(\mathbf{x},\hat{\mathbf{\Omega}}) = \frac{1}{4\pi}\cdot\sum_{g'=1}^{G}\Sigma_{s_0 g'\to g}(\mathbf{x})\cdot\phi_{g'}(\mathbf{x}) + \\ \frac{3\cdot\hat{\mathbf{\Omega}}}{4\pi}\cdot\sum_{g'=1}^{G}\Sigma_{s_1 g'\to g}(\mathbf{x})\cdot\mathbf{J}_{g'}(\mathbf{x}) + \frac{\chi_g}{4\pi}\sum_{g'=1}^{G}\nu\Sigma_{fg'}(\mathbf{x})\cdot\phi_{g'}(\mathbf{x}) + s_g(\mathbf{x},\hat{\mathbf{\Omega}})\end{aligned} \tag{3.7}$$

en función de los flujos angulares ψ_{mg} usando la definición 3.9 y explicitando el termino de la corriente como la integral del producto $\psi_{g'} \cdot \hat{\mathbf{\Omega}}$ según la definición 3.6

$$\begin{aligned}
&\hat{\mathbf{\Omega}} \cdot \operatorname{grad}\left[\psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\right] + \Sigma_{tg}(\mathbf{x}) \cdot \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}}) = \\
&\frac{1}{4\pi} \cdot \sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \cdot 4\pi \cdot \sum_{m'=1} w_{m'} \cdot \psi_{m'g'}(\mathbf{x}) + \\
&\frac{3 \cdot \hat{\mathbf{\Omega}}}{4\pi} \cdot \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \cdot \sum_{m'=1} \int_{\hat{\mathbf{\Omega}}_{m'}} \psi_{g'}(\mathbf{x}, \hat{\mathbf{\Omega}}) \cdot \hat{\mathbf{\Omega}}\, d\hat{\mathbf{\Omega}} + \\
&\frac{\chi_g}{4\pi} \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \cdot 4\pi \cdot \sum_{m'=1} w_{m'} \cdot \psi_{m'g'}(\mathbf{x}) + s_g(\mathbf{x}, \hat{\mathbf{\Omega}})
\end{aligned}$$

Ahora cancelamos los factores 4π en los términos de scattering lineal y fisión, integramos todos los términos con respecto a $\hat{\mathbf{\Omega}}$ sobre $\Delta\hat{\mathbf{\Omega}}_m$ y los analizamos uno por uno:

$$\begin{aligned}
&\underbrace{\int_{\hat{\mathbf{\Omega}}_m} \left\{\hat{\mathbf{\Omega}} \cdot \operatorname{grad}\left[\psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\right]\right\} d\hat{\mathbf{\Omega}}}_{\text{advección}} + \underbrace{\int_{\hat{\mathbf{\Omega}}_m} \left\{\Sigma_{tg}(\mathbf{x}) \cdot \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\right\} d\hat{\mathbf{\Omega}}}_{\text{absorción total}} = \\
&\underbrace{\int_{\hat{\mathbf{\Omega}}_m} \left\{\sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \cdot \sum_{m'=1} w_{m'} \cdot \psi_{m'g'}(\mathbf{x})\right\} d\hat{\mathbf{\Omega}}}_{\text{scattering isotrópico}} + \\
&\underbrace{\int_{\hat{\mathbf{\Omega}}_m} \left\{\frac{3 \cdot \hat{\mathbf{\Omega}}}{4\pi} \cdot \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \cdot \sum_{m'=1} \int_{\hat{\mathbf{\Omega}}_{m'}} \psi_{g'}(\mathbf{x}, \hat{\mathbf{\Omega}}') \cdot \hat{\mathbf{\Omega}}'\, d\hat{\mathbf{\Omega}}'\right\} d\hat{\mathbf{\Omega}}}_{\text{scattering linealmente anisótropo}} + \\
&\underbrace{\int_{\hat{\mathbf{\Omega}}_m} \left\{\chi_g \cdot \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \cdot \sum_{m'=1} w_{m'} \cdot \psi_{m'g'}(\mathbf{x})\right\} d\hat{\mathbf{\Omega}}}_{\text{fisión}} + \underbrace{\int_{\hat{\mathbf{\Omega}}_m} \left\{s_g(\mathbf{x}, \hat{\mathbf{\Omega}})\right\} d\hat{\mathbf{\Omega}}}_{\text{fuentes independientes}}
\end{aligned} \tag{3.9}$$

Comencemos por el termino de advección, revirtiendo el teorema 2.9

$$\begin{aligned}
\int_{\hat{\mathbf{\Omega}}_m} \left\{\hat{\mathbf{\Omega}} \cdot \operatorname{grad}\left[\psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\right]\right\} d\hat{\mathbf{\Omega}} &= \int_{\hat{\mathbf{\Omega}}_m} \operatorname{div}\left[\hat{\mathbf{\Omega}} \cdot \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\right] d\hat{\mathbf{\Omega}} \\
&= \operatorname{div}\left[\int_{\hat{\mathbf{\Omega}}_m} \hat{\mathbf{\Omega}} \cdot \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}}\right]
\end{aligned}$$

Prestemos atención a la integral. Supongamos que el flujo angular es constante a trozos[10] dentro de cada área $\Delta\hat{\mathbf{\Omega}}_m$. Entonces este valor constante es igual al valor medio de ψ en $\Delta\hat{\mathbf{\Omega}}_m$

[10] Del ingles *piecewise constant*.

$$\psi_g(\mathbf{x}, \hat{\mathbf{\Omega}}) = \left\langle \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}}) \right\rangle_{\hat{\mathbf{\Omega}}_m} = \psi_{mg}(\mathbf{x}) \quad \text{si } \hat{\mathbf{\Omega}} \in \Delta\hat{\mathbf{\Omega}}_m$$

En estas condiciones, ψ puede salir de la integral

$$\int_{\hat{\mathbf{\Omega}}_m} \hat{\mathbf{\Omega}} \cdot \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}} \approx \psi_{mg}(\mathbf{x}) \cdot \int_{\hat{\mathbf{\Omega}}_m} \hat{\mathbf{\Omega}}\, d\hat{\mathbf{\Omega}}$$

Definición 3.10. Llamamos $\hat{\mathbf{\Omega}}_m$ a la dirección que resulta ser el valor medio $\left\langle \hat{\mathbf{\Omega}} \right\rangle_{\hat{\mathbf{\Omega}}_m}$ de todas las direcciones integradas en el área $\Delta\hat{\mathbf{\Omega}}_m$ sobre la esfera unitaria, es decir

$$\hat{\mathbf{\Omega}}_m = \left\langle \hat{\mathbf{\Omega}} \right\rangle_{\hat{\mathbf{\Omega}}_m} = \frac{\displaystyle\int_{\Delta\hat{\mathbf{\Omega}}_m} \hat{\mathbf{\Omega}}\, d\hat{\mathbf{\Omega}}}{\displaystyle\int_{\Delta\hat{\mathbf{\Omega}}_m} d\hat{\mathbf{\Omega}}}$$

Corolario 3.3. *La integral del versor $\hat{\mathbf{\Omega}}$ sobre la fracción $\Delta\hat{\mathbf{\Omega}}_m$ de la esfera unitaria es igual al producto de $\hat{\mathbf{\Omega}}_m$ por $\Delta\hat{\mathbf{\Omega}}_m$:*

$$\int_{\Delta\hat{\mathbf{\Omega}}_m} \hat{\mathbf{\Omega}}\, d\hat{\mathbf{\Omega}} = \hat{\mathbf{\Omega}}_m \cdot \Delta\hat{\mathbf{\Omega}}_m$$

Corolario 3.4. *La integral del producto del versor $\hat{\mathbf{\Omega}}$ con el flujo angular del grupo g sobre $\Delta\hat{\mathbf{\Omega}}_m$ es aproximadamente igual al producto $\psi_{mg} \cdot \Delta\hat{\mathbf{\Omega}}_m$:*

$$\int_{\hat{\mathbf{\Omega}}_m} \hat{\mathbf{\Omega}} \cdot \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}} \approx \hat{\mathbf{\Omega}}_m \cdot \psi_{mg}(\mathbf{x}) \cdot \Delta\hat{\mathbf{\Omega}}_m$$

Podemos volver a aplicar el teorema 2.9 para escribir el término de advección como

$$\begin{aligned} \operatorname{div}\left[\int_{\hat{\mathbf{\Omega}}_m} \hat{\mathbf{\Omega}} \cdot \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}}\right] &\approx \operatorname{div}\left[\hat{\mathbf{\Omega}}_m \cdot \psi_{mg}(\mathbf{x}) \cdot \Delta\hat{\mathbf{\Omega}}_m\right] \\ &\approx \hat{\mathbf{\Omega}}_m \cdot \operatorname{grad}\left[\psi_{mg}(\mathbf{x})\right] \cdot \Delta\hat{\mathbf{\Omega}}_m \end{aligned} \tag{3.10}$$

Pasemos ahora al término de absorciones totales de la ecuación 3.9. La sección eficaz no depende de $\hat{\mathbf{\Omega}}$ por lo que puede salir fuera de la integral

$$\int_{\hat{\mathbf{\Omega}}_m} \left\{\Sigma_{tg}(\mathbf{x}) \cdot \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\right\}\, d\hat{\mathbf{\Omega}} = \Sigma_{tg}(\mathbf{x}) \cdot \int_{\hat{\mathbf{\Omega}}_m} \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}}$$

Por el corolario 3.1 la última integral es $\psi_{mg} \cdot \Delta\hat{\mathbf{\Omega}}_m$, entonces

$$\int_{\hat{\mathbf{\Omega}}_m} \left[\Sigma_{tg}(\mathbf{x}) \cdot \psi_g(\mathbf{x}, \hat{\mathbf{\Omega}})\right] \, d\hat{\mathbf{\Omega}} = \left[\Sigma_{tg}(\mathbf{x}) \cdot \psi_{mg}(\mathbf{x})\right] \cdot \Delta\hat{\mathbf{\Omega}}_m \tag{3.11}$$

El término de scattering isotrópico queda

$$\int_{\hat{\mathbf{\Omega}}_m} \sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \cdot \sum_{m'=1} w_{m'} \cdot \psi_{m'g'}(\mathbf{x}) \, d\hat{\mathbf{\Omega}} = \left[\sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \cdot \sum_{m'=1} w_{m'} \cdot \psi_{m'g'}(\mathbf{x})\right] \cdot \Delta\hat{\mathbf{\Omega}}_m \tag{3.12}$$

ya que el integrando no depende de $\hat{\mathbf{\Omega}}$.

El integrando del término de scattering linealmente anisótropo sí depende de $\hat{\mathbf{\Omega}}$.

$$\int_{\hat{\mathbf{\Omega}}_m} \left[\frac{3 \cdot \hat{\mathbf{\Omega}}}{4\pi} \cdot \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \cdot \sum_{m'=1} \int_{\hat{\mathbf{\Omega}}_{m'}} \psi_{g'}(\mathbf{x}, \hat{\mathbf{\Omega}}') \cdot \hat{\mathbf{\Omega}}' \, d\hat{\mathbf{\Omega}}'\right] d\hat{\mathbf{\Omega}}$$

Primero notamos que el corolario 3.4 nos indica, de manera aproximada, el resultado de la integral sobre $\hat{\mathbf{\Omega}}_{m'}$: $\hat{\mathbf{\Omega}}_{m'} \psi_{m'g'} \Delta\hat{\mathbf{\Omega}}_{m'}$. A su vez, $\Delta\hat{\mathbf{\Omega}}_{m'} = 4\pi w_{m'}$, por lo que

$$\int_{\hat{\mathbf{\Omega}}_m} \left[\frac{3 \cdot \hat{\mathbf{\Omega}}}{4\pi} \cdot \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \cdot \sum_{m'=1} 4\pi \cdot w_{m'} \cdot \psi_{m'g'}(\mathbf{x}) \cdot \hat{\mathbf{\Omega}}_{m'}\right] d\hat{\mathbf{\Omega}} =$$
$$3 \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \cdot \sum_{m'=1} w_{m'} \cdot \psi_{m'g'}(\mathbf{x}) \cdot \hat{\mathbf{\Omega}}_{m'} \int_{\hat{\mathbf{\Omega}}_m} \hat{\mathbf{\Omega}} \, d\hat{\mathbf{\Omega}}$$

Una vez mas, la integral sobre $\hat{\mathbf{\Omega}}_m$ ya la hemos resuelto (exactamente) en el corolario 3.3, y es igual a $\hat{\mathbf{\Omega}}_m \cdot \Delta\hat{\mathbf{\Omega}}_m$. Entonces el término de scattering linealmente anisótropo es aproximadamente igual a

$$\int_{\hat{\mathbf{\Omega}}_m} \left[\frac{3 \cdot \hat{\mathbf{\Omega}}}{4\pi} \cdot \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \cdot \sum_{m'=1} \int_{\hat{\mathbf{\Omega}}_{m'}} \psi_{g'}(\mathbf{x}, \hat{\mathbf{\Omega}}') \cdot \hat{\mathbf{\Omega}}' \, d\hat{\mathbf{\Omega}}'\right] d\hat{\mathbf{\Omega}} \approx$$
$$\left[3 \cdot \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \cdot \sum_{m'=1} w_{m'} \cdot \left(\hat{\mathbf{\Omega}}_m \cdot \hat{\mathbf{\Omega}}_{m'}\right) \cdot \psi_{m'g'}(\mathbf{x})\right] \cdot \Delta\hat{\mathbf{\Omega}}_m \tag{3.13}$$

El término de fisiones es similar al de scattering isotrópico en el sentido de que el integrando no depende de $\hat{\mathbf{\Omega}}$ entonces su integral sobre $\Delta\hat{\mathbf{\Omega}}_m$ es directamente

$$\int_{\hat{\mathbf{\Omega}}_m} \chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \cdot \sum_{m'=1} w_{m'} \cdot \psi_{m'g'}(\mathbf{x}) \, d\hat{\mathbf{\Omega}} = \left[\chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \cdot \sum_{m'=1} w_{m'} \cdot \psi_{m'g'}(\mathbf{x})\right] \cdot \Delta\hat{\mathbf{\Omega}}_m \tag{3.14}$$

Para completar el análisis de la ecuación 3.9, en el término de las fuentes independientes usamos el concepto de valor medio

$$\int_{\hat{\mathbf{\Omega}}_m} s_g(\mathbf{x}, \hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}} = \left\langle s_g(\mathbf{x}, \hat{\mathbf{\Omega}}) \right\rangle_{\hat{\mathbf{\Omega}}_m} \cdot \Delta\hat{\mathbf{\Omega}}$$

Definición 3.11. En forma análoga a la definición 3.9, definimos a la fuente independiente del grupo g en la dirección m como

$$s_{mg}(\mathbf{x}) = \left\langle s(\mathbf{x}, \hat{\mathbf{\Omega}}) \right\rangle_{\hat{\mathbf{\Omega}}_m} = \frac{\displaystyle\int_{\Delta\hat{\mathbf{\Omega}}_m} s_g(\mathbf{x}, \hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}}}{\displaystyle\int_{\Delta\hat{\mathbf{\Omega}}_m} d\hat{\mathbf{\Omega}}}$$

El término de fuentes independientes es entonces

$$\int_{\hat{\mathbf{\Omega}}_m} s_g(\mathbf{x}, \hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}} = s_{mg}(\mathbf{x}) \cdot \Delta\hat{\mathbf{\Omega}}_m \tag{3.15}$$

Juntemos ahora las ecuaciones

- 3.10 (advección)
- 3.11 (absorciones)
- 3.12 (scattering isotrópico)
- 3.13 (scattering linealmente anisótropo)
- 3.14 (fisiones)
- 3.15 (fuentes independientes)

para re-escribir la ecuación 3.9 como

$$\begin{gathered}
\left[\hat{\mathbf{\Omega}}_m \cdot \operatorname{grad}\left[\psi_{mg}(\mathbf{x})\right]\right] \cdot \Delta\hat{\mathbf{\Omega}}_m + \left[\Sigma_{tg}(\mathbf{x}) \cdot \psi_{mg}(\mathbf{x})\right] \cdot \Delta\hat{\mathbf{\Omega}}_m = \\
\left[\sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \cdot \sum_{m'=1} w_{m'} \cdot \psi_{m'g'}(\mathbf{x})\right] \cdot \Delta\hat{\mathbf{\Omega}}_m + \\
\left[3 \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \cdot \sum_{m'=1} w_{m'} \cdot \left(\hat{\mathbf{\Omega}}_m \cdot \hat{\mathbf{\Omega}}_{m'}\right) \cdot \psi_{m'g'}(\mathbf{x})\right] \cdot \Delta\hat{\mathbf{\Omega}}_m + \\
\left[\chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \cdot \sum_{m'=1} w_{m'} \cdot \psi_{m'g'}(\mathbf{x})\right] \cdot \Delta\hat{\mathbf{\Omega}}_m + s_{mg}(\mathbf{x}) \cdot \Delta\hat{\mathbf{\Omega}}_m
\end{gathered}$$

Dividiendo ambos miembros por $\Delta\hat{\mathbf{\Omega}}$ obtenemos las MG ecuaciones diferenciales de transporte en G grupos de energías y M direcciones angulares, según la discretización angular denominada en la literatura "ordenadas discretas"

$$\hat{\mathbf{\Omega}}_m \cdot \text{grad}\left[\psi_{mg}(\mathbf{x})\right] + \Sigma_{tg}(\mathbf{x}) \cdot \psi_{mg}(\mathbf{x}) = \sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \sum_{m'=1} w_{m'} \psi_{m'g'}(\mathbf{x}) +$$

$$3 \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \sum_{m'=1} w_{m'} \left(\hat{\mathbf{\Omega}}_m \cdot \hat{\mathbf{\Omega}}_{m'}\right) \psi_{m'g'}(\mathbf{x}) + \chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \sum_{m'=1} w_{m'} \psi_{m'g'}(\mathbf{x}) + s_{mg}(\mathbf{x}) \tag{3.16}$$

Observación. El único operador diferencial que aparece en la ecuación 3.16 es el gradiente espacial del flujo angular ψ_{mg} del grupo g en la dirección m en el término de advección.

Observación. Todos los operadores integrales que estaban presentes en la ecuación 2.41 han sido reemplazados por sumatorias finitas.

Observación. La única aproximación numérica que tuvimos que hacer para obtener la ecuación 3.16 a partir de la ecuación 3.7 fue suponer que el flujo angular ψ_g es uniforme a trozos en cada segmento de área $\Delta\hat{\mathbf{\Omega}}_m$ en los términos de

a. absorciones totales (ecuación 3.11), y
b. scattering linealmente anisótropo (ecuación 3.13).

Por ejemplo, la figura 3.2 ilustra un caso en el que cada octante de la esfera unitaria está dividido en tres áreas iguales, dando lugar a $M = 3 \times 8 = 24$ direcciones. En cada una de las áreas mostradas, asumimos que el flujo angular $\psi(\mathbf{x}, \hat{\mathbf{\Omega}})$ es uniformemente igual a $\psi_{mg}(\mathbf{x})$, siendo $\mathbf{x}$ en este caso la posición del centro de la esfera unidad. Esta suposición es usual en los esquemas basados en el método de volúmenes finitos.

Observación. El esquema numérico es consistente ya que en el límite $\Delta\hat{\mathbf{\Omega}}_m \to d\hat{\mathbf{\Omega}}_m$ la suposición es exacta y el operador discretizado coincide con el operador continuo.

3.3.1. Conjuntos de cuadratura

Para completar el método de las ordenadas discretas debemos especificar M pares de direcciones y pesos $(\hat{\mathbf{\Omega}}_m, w_m)$ para $m = 1, \dots, M$. En tres dimensiones, si utilizamos M direcciones tales que

$$M = N \cdot (N + 2) \tag{3.17}$$

decimos que estamos implementando el método de ordenadas discretas S_N. Esta relación numérica entre N y M es histórica y en esta tesis la mantenemos.

Las direcciones $\hat{\mathbf{\Omega}}_m = [\hat{\Omega}_{mx}\, \hat{\Omega}_{my}\, \hat{\Omega}_{mz}]^T$ deben ser versores unitarios tales que

$$\hat{\Omega}_{mx}^2 + \hat{\Omega}_{my}^2 + \hat{\Omega}_{mz}^2 = 1 \tag{3.18}$$

y para poder aplicar el teorema 3.2, los pesos w_m deben estar normalizados a uno, es decir

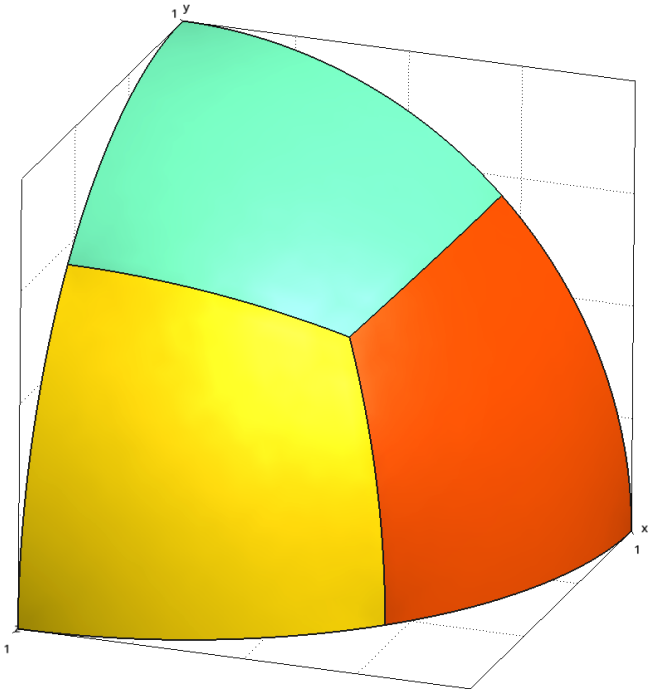


Figura 3.2.: Una partición de la esfera unidad en 24 fracciones de área, todas iguales entre sí. La suposición central de la derivación del método S_N realizado en esta tesis es que en cada una de éstas áreas el flujo angular es constante. Se muestra sólo el primero de los ocho octantes.

$$\sum_{m=1}^{M} w_m = 1$$

Existen varias maneras de elegir los M pares de forma tal de cumplir estas dos condiciones. En primer lugar, para poder poner condiciones de contorno de simetría en planos paralelos a los tres planos coordinados x-y, x-z e y-z requerimos que si la dirección $\hat{\mathbf{\Omega}} = [\hat{\Omega}_x\, \hat{\Omega}_y\, \hat{\Omega}_z]^T$ con $\hat{\Omega}_x > 0$, $\hat{\Omega}_y > 0$ y $\hat{\Omega}_z > 0$ pertenece al conjunto de cuadratura, entonces también tienen que estar las siguientes siete direcciones

$$\begin{bmatrix} +\hat{\Omega}_x & +\hat{\Omega}_y & -\hat{\Omega}_z \end{bmatrix}^T$$
$$\begin{bmatrix} +\hat{\Omega}_x & -\hat{\Omega}_y & +\hat{\Omega}_z \end{bmatrix}^T$$
$$\begin{bmatrix} +\hat{\Omega}_x & -\hat{\Omega}_y & -\hat{\Omega}_z \end{bmatrix}^T$$
$$\begin{bmatrix} -\hat{\Omega}_x & +\hat{\Omega}_y & +\hat{\Omega}_z \end{bmatrix}^T$$
$$\begin{bmatrix} -\hat{\Omega}_x & +\hat{\Omega}_y & -\hat{\Omega}_z \end{bmatrix}^T$$
$$\begin{bmatrix} -\hat{\Omega}_x & -\hat{\Omega}_y & +\hat{\Omega}_z \end{bmatrix}^T$$
$$\begin{bmatrix} -\hat{\Omega}_x & -\hat{\Omega}_y & -\hat{\Omega}_z \end{bmatrix}^T$$

Luego es suficiente definir las $N(N+2)/8$ direcciones del primero de los ocho octantes y luego permutar los signos para obtener las direcciones correspondientes a los otros siete octantes. En este trabajo utilizamos la cuadratura de nivel simétrico [34] o de simetría completa [58] en la que las direcciones son simétricas en cada octante. Consiste en tomar tres cosenos directores μ_i, μ_j y μ_k de un conjunto de $N/2$ valores positivos y permutarlos de todas las maneras posibles para obtener $N(N+2)/8$ combinaciones como ilustramos en la figura 3.3 y continuamos discutiendo a continuación.

Teorema 3.3. *En la cuadratura de nivel simétrico, no todos los $N/2$ posibles cosenos directores son independientes. Para S_2 hay una única dirección posible. Para $N > 2$ sólo uno de los cosenos directores es independiente. El resto de los valores depende del primero.*

Demostración. Para $N = 2$ hay una única dirección posible en cada octante que proviene de un único cosenos director μ_1 ya que $N/2 = 1$. Luego $\hat{\mathbf{\Omega}}_1 = [\mu_1\ \mu_1\ \mu_1]^T$. Para preservar la condición de normalización, debe ser $\mu_1 = 1/\sqrt{3}$.

Para $N > 2$, sean $\mu_1 \leq \mu_2 \leq \cdots < \mu_{N/2}$ los posibles cosenos directores del conjunto. Supongamos que para la dirección m tenemos $\hat{\Omega}_{mx} = \mu_i$, $\hat{\Omega}_{my} = \mu_j$ y $\hat{\Omega}_{mz} = \mu_k$. Entonces, por el requerimiento de normalización de la ecuación 3.18 debemos tener

$$\mu_i^2 + \mu_j^2 + \mu_k^2 = 1 \tag{3.19}$$

Tomemos ahora otra dirección diferente m' pero manteniendo el primer componente $\hat{\Omega}_{m'x} = \mu_i$ y haciendo que $\hat{\Omega}_{m'y} = \mu_{j+1}$. Para poder satisfacer la ecuación 3.19, debido a que $\mu_{j+1} > \mu_j$ entonces $\hat{\Omega}_{m'z} = \mu_{k-1}$ ya que $\mu_{k-1} < \mu_k$. Entonces

$$\mu_i^2 + \mu_{j+1}^2 + \mu_{k-1}^2 = 1 \tag{3.20}$$

De las ecuaciones 3.19 y 3.20 obtenemos

$$\mu_{j+1}^2 - \mu_j = \mu_k^2 - \mu_{k-1}^2$$

Como esta condición debe cumplirse para todo j y para todo k, entonces

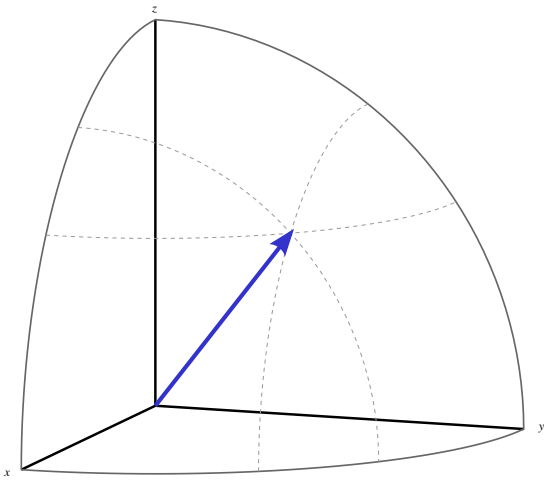
$$\mu_i^2 = \mu_{i-1} + C$$

para todo $1 < i \leq N/2$, con C una constante a determinar. Luego el i-ésimo coseno director es

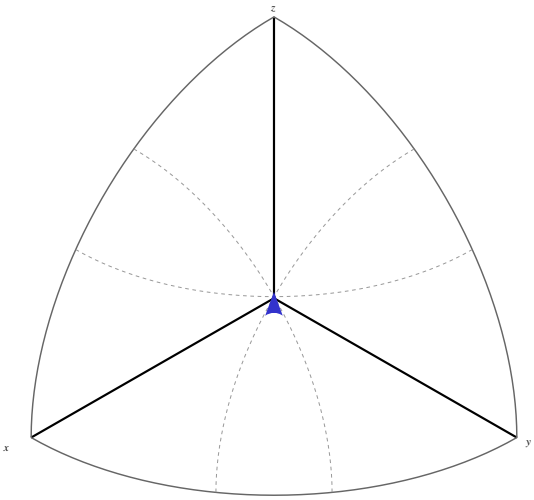
$$\mu_i^2 = \mu_1 + C \cdot (i-1)$$

Si tomamos $\hat{\Omega}_{mx} = \hat{\Omega}_{my} = \mu_1$ y $\hat{\Omega}_{mz} = \mu_{N/2}$, por la condición de magnitud unitaria debe ser

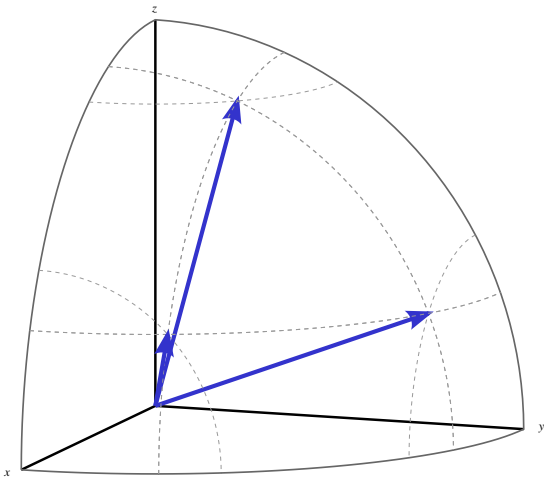
$$2\mu_1^2 + \mu_{N/2}^2 = 1$$



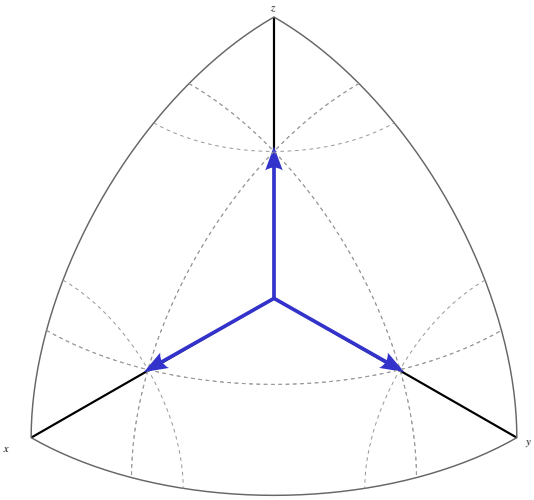
(a) S_2



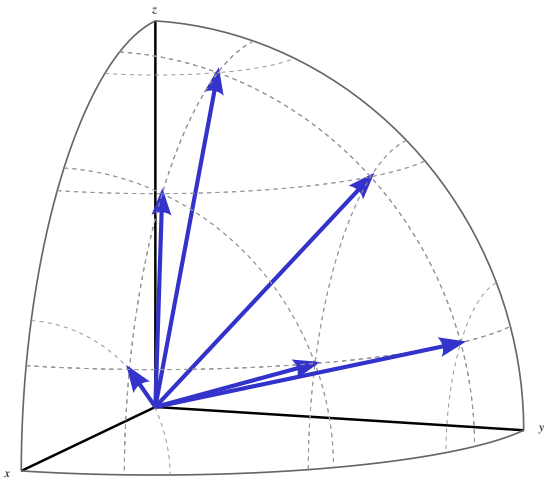
(b) S_2



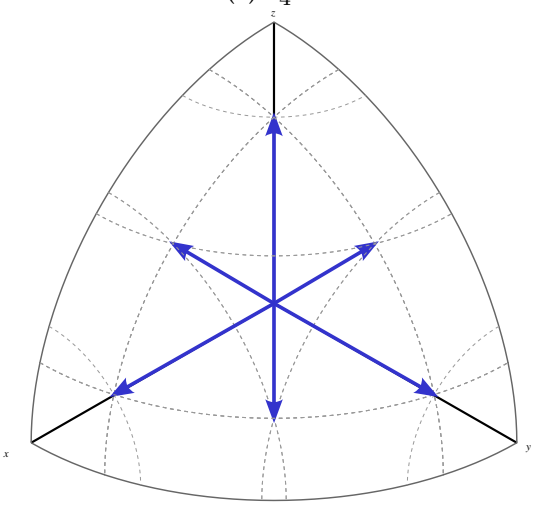
(c) S_4



(d) S_4



(e) S_6



(f) S_6

Figura 3.3.: Direcciones de cuadratura de nivel simétrico para S_2, S_4 y S_6 en el primer cuadrante.

de donde podemos determinar la constante C como

$$C = \frac{2 \cdot (1 - 3\mu_1^2)}{N - 2}$$

Finalmente, una vez seleccionado el coseno director μ_1, podemos calcular el resto de los $N/2 - 1$ valores como

$$\mu_i = \sqrt{\mu_1^2 + (2 - 6\mu_1^2) \cdot \frac{(i-1)}{N-2}}$$

para $i = 2, \dots, N/2$.

□

S_2	S_4
1	1 1 1
S_6	S_8
1 2 2 1 2 1	1 2 2 2 3 2 1 2 2 1
S_{10}	S_{12}
1 2 2 3 4 3 2 4 4 2 1 2 3 2 1	1 2 2 3 4 3 3 5 5 3 2 4 5 4 2 1 2 3 3 2 1
S_{14}	S_{16}
1 2 2 3 5 3 4 6 6 4 3 6 7 6 3 2 5 6 6 5 2 1 2 3 4 3 2 1	1 2 2 3 5 3 4 6 6 4 4 7 8 7 4 3 6 8 8 6 3 2 5 6 7 6 5 2 1 2 3 4 4 3 2 1

Tabla 3.1.: Triángulos de cuadratura de nivel simétrico indicando los puntos de igual peso.[11] Las direcciones con el mismo número tienen el mismo peso. Estos triángulos fueron generados por la herramienta descripta en el capítulo 4 por lo que es posible generar esta tabla para un N arbitrario.

[11] Equivalentes a la figura 6.4 de la página 205 de [58] y a la figura 4.3 de la página 161 de [34].

Observación. Si el primer coseno director μ_1 es cercano a cero, las direcciones tienden a formar un cluster alrededor de los polos. Si el primer coseno director μ_1 es cercano a $1/\sqrt{3}$, las direcciones tienden a formar un cluster alrededor del centro de cada octante.

Si miráramos el octante desde la dirección $[1/\sqrt{3}\ 1/\sqrt{3}\ 1/\sqrt{3}]^T$ como en la segunda columna de la figura 3.3 y le asignáramos el mismo entero a cada dirección que sea una permutación de los mismos tres cosenos directores, veríamos lo que indica la tabla 3.1. Las condiciones de simetría requieren que los pesos w_m y $w_{m'}$ asociados a dos direcciones $\hat{\Omega}_m$ y $\hat{\Omega}_{m'}$ cuyos cosenos directores son permutaciones entre sí deban ser iguales. Por lo tanto, los enteros de la tabla 3.1 terminan indicando el índice del peso a utilizar.

Observación. Para $N = 2$ el triángulo de la tabla 3.1 tiene $N/2 = 1$ fila. Para cada nuevo N, se agrega una fila con $N/2$ nuevas direcciones. Entonces la cantidad de direcciones en un octante para S_N es

$$\frac{\frac{N}{2}\left(\frac{N}{2}+1\right)}{2} = \frac{\frac{1}{2}\cdot N\cdot\frac{1}{2}\left(N+2\right)}{2} = \frac{N\cdot(N+2)}{8}$$

de donde sigue la ecuación 3.17.

La elección de los w_m debe ser tal que la cuadratura

$$\int_{4\pi} f(\hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}} \approx 4\pi \cdot \sum_{w=1}^{M} w_m \cdot \left\langle f(\hat{\mathbf{\Omega}})\right\rangle_m$$

del teorema teorema 3.2 arroje los resultados más precisos posibles en la ecuación de transporte de neutrones. En este sentido, dos condiciones importantes son las siguientes.

1. Dado que la corriente neta se aproxima como

$$\mathbf{J}_g(\mathbf{x}) \approx \sum_{m=1}^{M} w_m \cdot \hat{\mathbf{\Omega}}_m \cdot \phi_{mg}(\mathbf{x})$$

   entonces para poder recuperar una corriente neta igual a cero para un flujo angular uniforme debemos tener

$$\sum_{m=1}^{M} w_m \cdot \hat{\mathbf{\Omega}}_m \cdot = 0$$

2. Para poder recuperar el resultado

$$\psi_{mg}(\mathbf{x}) \approx \frac{1}{4\pi}\phi_{mg}(\mathbf{x}) + 3\cdot\hat{\mathbf{\Omega}}_m\cdot\mathbf{J}_{mg}(\mathbf{x})$$

   entonces

$$\sum_{m=1}^{M} w_m \cdot \hat{\mathbf{\Omega}}_m^2 \cdot = \frac{1}{3}$$

Observación. Para extender las $N(N+2)/8$ direcciones a los demás cuadrantes, podemos notar que si asignamos un índice n a cada uno de los ocho octantes de la siguiente manera:

0. $x > 0, y > 0, z > 0$
1. $x < 0, y > 0, z > 0$
2. $x > 0, y < 0, z > 0$
3. $x < 0, y < 0, z > 0$
4. $x > 0, y > 0, z < 0$
5. $x < 0, y > 0, z < 0$
6. $x > 0, y < 0, z < 0$
7. $x < 0, y < 0, z < 0$

entonces el desarrollo binario del índice n tiene tres bits y éstos indican si hubo un cambio de signo o no en cada uno de los tres ejes con respecto al primer cuadrante, que corresponde a $n = 0$. De esta manera, es posible generar las direcciones $\hat{\Omega}_m$ para $m = N(N+2)/8+1, N(N+2)$ a partir de las direcciones del primer cuadrante $\hat{\Omega}_j$ para $j = 1, N(N+2)/8$ con el siguiente algoritmo

for $c = 1, \dots, 7$ **do**
  for $m = 1, \dots, N(N+2)/8$ **do**
    $\Omega_{c\cdot N(N+2)/8+m\,,\,x} \leftarrow [(c\&1)?(-1) : (+1)] \cdot \Omega_{m\,,\,x}$
    $\Omega_{c\cdot N(N+2)/8+m\,,\,y} \leftarrow [(c\&2)?(-1) : (+1)] \cdot \Omega_{m\,,\,y}$
    $\Omega_{c\cdot N(N+2)/8+m\,,\,z} \leftarrow [(c\&4)?(-1) : (+1)] \cdot \Omega_{m\,,\,z}$
    $w_{c\cdot N(N+2)/8+m} = w_j$
  end
end

Algoritmo 1: Extensión del primer octante a los otros siete

donde

- el símbolo "et" `&` indica el operador binario `AND` y
- el signo de pregunta `?` el operador ternario de decisión.

El cálculo detallado de los pesos está fuera del alcance de esta tesis. La herramienta computacional tiene cargados el primer coseno director de cada N y los pesos reportados en las referencias [34], [58]. Consultar el código fuente de la implementación descripta en el capítulo 4 para ver los detalles algorítmicos y numéricos.

3.3.1.1. Dos dimensiones

El caso bidimensional en realidad es un problema en tres dimensiones pero sin dependencia de los parámetros del problema en una de las variables espaciales, digamos z. De esta manera, el dominio $U \in \mathbb{R}^2$ de la geometría está definido sólo sobre el plano x-y y las direcciones de vuelo $\hat{\mathbf{\Omega}}$ de los neutrones son simétricas con respecto a este plano ya que por cada dirección $\hat{\mathbf{\Omega}} = [\hat{\Omega}_x\, \hat{\Omega}_y\, \hat{\Omega}_z]$ con $\hat{\Omega}_z > 0$ hay una dirección simétrica $\hat{\mathbf{\Omega}}' = [\hat{\Omega}_x\, \hat{\Omega}_y\, -\hat{\Omega}_z]$ (figura 3.4). Luego, las posibles direcciones se reducen a la mitad, es decir $N(N+2)/2$.

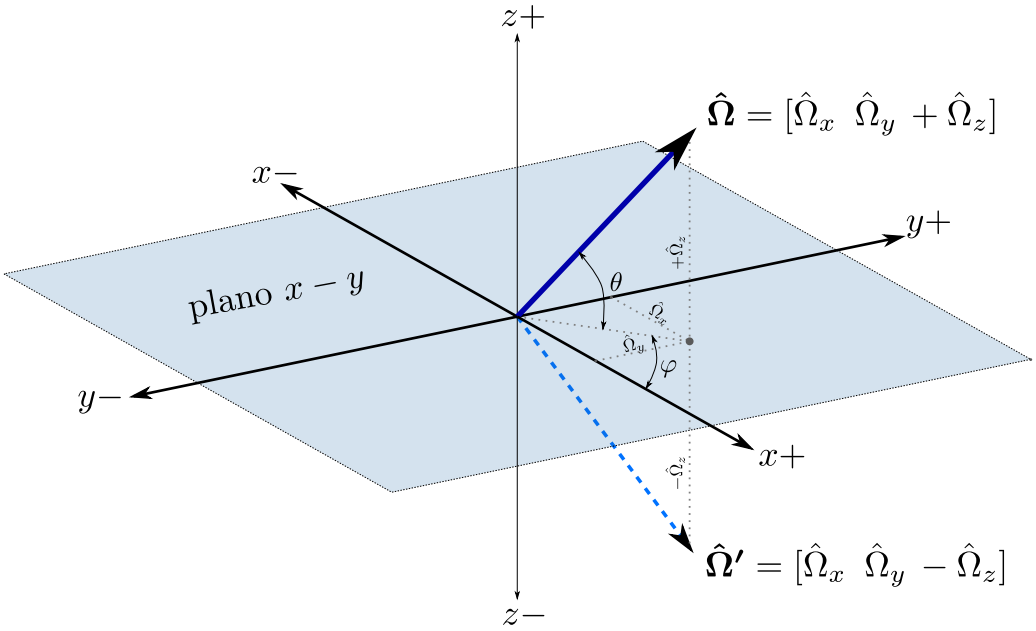


Figura 3.4.: Simetría con respecto al plano x-y en un problema bi-dimensional. Por cada dirección $\hat{\mathbf{\Omega}}$ con $\hat{\Omega}_z > 0$ (línea llena) hay una dirección $\hat{\mathbf{\Omega}}'$ simétrica e igualmente posible con $\hat{\Omega}_z < 0$ (línea de trazos).

Como la derivada espacial del flujo angular con respecto a z es cero entonces por un lado podemos escribir el término de transporte en la ecuación 3.16 como

$$\hat{\Omega}_{mx} \cdot \frac{\partial \psi_{mg}(x,y)}{\partial x} + \hat{\Omega}_{my} \cdot \frac{\partial \psi_{mg}(x,y)}{\partial y}$$

donde ahora $m = 1, \ldots, M = N(N+2)/2$. La componente Ω_{mz} no aparece explícitamente en las ecuaciones pero sí lo hace implícitamente en la elección de las direcciones, ya que sigue siendo válida la discusión de la sección anterior. Esto implica que en cada cuadrante tenemos nuevamente $N(N+2)/8$ direcciones posibles, que luego debemos rotar para obtener las M direcciones en los cuatro cuadrantes. Dado que por un lado los pesos deben estar normalizados a uno y por otro para cada dirección con $\hat{\Omega}_z > 0$ hay otra dirección simétrica con $\hat{\Omega}_z < 0$, entonces el conjunto de cuadraturas de nivel simétrico para el primer cuadrante de un dominio de dos dimensiones consiste en las mismas $N(N+2)/8$ direcciones correspondientes a tres dimensiones.

3.3.1.2. Una dimensión

El caso unidimensional es radicalmente diferente a los otros dos. Si tomamos al eje x como la dirección de dependencia espacial, las posibles direcciones de viaje pueden depender sólo del ángulo cenital θ ya que la simetría implica que todas las posibles direcciones azimutales con respecto al eje x son igualmente posibles.

El término de transporte es ahora entonces

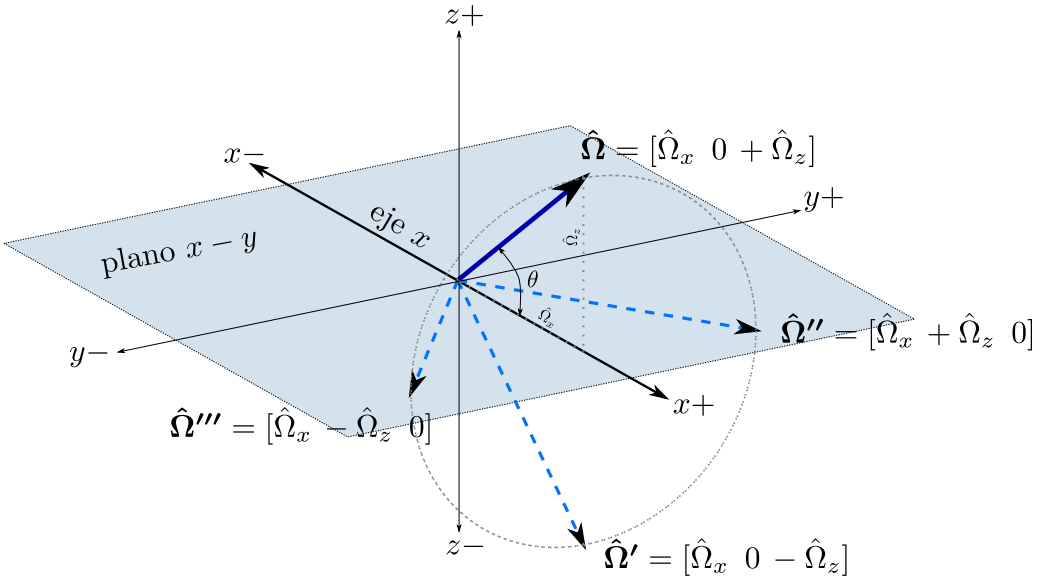


Figura 3.5.: Simetría con respecto al eje~x en un problema unidimensional. Por cada dirección~$\hat{\mathbf{\Omega}}$ (línea llena) hay infinitas direcciones simétricas e igualmente posibles apuntando en la dirección del círculo subtendido por el ángulo~$\theta = \arctan(\hat{\Omega}_z/\hat{\Omega}_x)$, representadas por las tres direcciones primadas (líneas de trazos).

$$\hat{\Omega}_{mx} \cdot \frac{\partial \psi_{mg}(x)}{\partial x}$$

El hecho de que no una sino dos componentes de $\hat{\mathbf{\Omega}}$ no aparezcan explícitamente relaja mucho más las condiciones para la elección de las $M = N$ direcciones. En efecto, la única condición es simetría completa entre el semieje $x > 0$ y el semieje $x < 0$, lo que nos deja con $N/2$ direcciones en cada semieje, todas ellas libres e independientes.

Para seleccionar las $N/2$ direcciones y sus pesos asociados, notamos que en una dimensión

$$\int_{4\pi} f(\hat{\mathbf{\Omega}})\, d\hat{\mathbf{\Omega}} = 2\pi \int_{-1}^{1} f(\hat{\Omega}_x)\, d\hat{\Omega}_x \simeq 2\pi \sum_{m=1}^{N} w_m \cdot f_m = 4\pi \sum_{m=1}^{N} \frac{w_m}{2} \cdot f_m = 4\pi \sum_{m=1}^{N} w_m \cdot f_m \tag{3.21}$$

Si los puntos $\hat{\Omega}_{xm}$ y los pesos $w_m = 2 \cdot w_m$ son los asociados a la integración de Gauss y $f(\hat{\Omega}_x)$ es un polinomio de orden $2N - 1$ o menos, entonces la integración es exacta. En la tabla tabla 3.2 mostramos el conjunto de cuadraturas utilizadas para una dimensión, que contiene esencialmente las abscisas y los pesos de la cuadratura de Gauss.

3.4. Discretización en espacio

Hasta el momento, tenemos por un lado las G ecuaciones de difusión multigrupo

	m	$\hat{\Omega}_{mx}$	$2 \cdot w_m$
S_2	1	$\sqrt{\frac{1}{3}}$	1
S_4	1	$\sqrt{\frac{3}{7} - \frac{2}{7}\sqrt{\frac{6}{5}}}$	0.6521451549
	2	$\sqrt{\frac{3}{7} + \frac{2}{7}\sqrt{\frac{6}{5}}}$	0.3478548451
S_6	1	0.2386191860	0.4679139346
	2	0.6612093864	0.3607615730
	3	0.9324695142	0.1713244924
S_8	1	0.1834346424	0.3626837834
	2	0.5255324099	0.5255324099
	3	0.7966664774	0.2223810344
	4	0.9602898564	0.1012285363

(a)

Tabla 3.2.: Conjuntos de cuadratura para problemas unidimensionales. Las direcciones $\hat{\Omega}_{mx}$ coinciden con las abscisas de la cuadratura de Gauss. Los pesos w_m de ordenadas discretas son la mitad de los pesos w_m de la cuadratura de Gauss. Las direcciones $m = N/2+1, \dots, N$ no se muestran pero se obtienen como $\hat{\Omega}_{N/2+m\,x} = -\hat{\Omega}_{mx}$ y $w_{N/2+m} = w_m$.

$$-\text{div}\Big[D_g(\mathbf{x}) \cdot \text{grad}\left[\phi_g(\mathbf{x})\right]\Big] + \Sigma_{tg}(\mathbf{x}) \cdot \phi_g(\mathbf{x}) = \sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \cdot \phi_{g'}(\mathbf{x}) + \chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \cdot \phi_{g'}(\mathbf{x}) + s_{0g}(\mathbf{x}) \tag{3.8}$$

y las MG ecuaciones de transporte S_N multigrupo

$$\hat{\mathbf{\Omega}}_m \cdot \text{grad}\left[\psi_{mg}(\mathbf{x})\right] + \Sigma_{tg}(\mathbf{x}) \cdot \psi_{mg}(\mathbf{x}) = \sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \sum_{m'=1} w_{m'} \psi_{m'g'}(\mathbf{x}) +$$

$$3 \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \sum_{m'=1} w_{m'} \left(\hat{\mathbf{\Omega}}_m \cdot \hat{\mathbf{\Omega}}_{m'}\right) \psi_{m'g'}(\mathbf{x}) + \chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \sum_{m'=1} w_{m'} \psi_{m'g'}(\mathbf{x}) + s_{mg}(\mathbf{x}) \tag{3.16}$$

en las que las incógnitas ϕ_g y ψ_{mg} dependen solamente del espacio $\mathbf{x}$. En esta sección empleamos el método de elementos finitos [91] para discretizar la variable independiente espacial y obtener finalmente un sistema de ecuaciones algebraicas que nos permita resolver neutrónica a nivel de núcleo en forma numérica con una (o más) computadora(s) digital(es).

Existe una gran cantidad de teoría matemática detrás del método de elementos finitos para resolver ecuaciones diferenciales a partir de formulaciones débiles o variacionales. Esencialmente el grueso de la literatura teórica [12], [26], [44] se centra en probar

1. que la formulación débil (definición 3.13) de una ecuación diferencial es formalmente correcta con respecto a derivabilidad e integrabilidad en el sentido de distribuciones sobre espacios de Hilbert,
2. que soluciones continuas pero no necesariamente diferenciables en a lo más un sub-espacio de medida cero tienen sentido matemático, y
3. que el esquema numérico es consistente (definición 3.2), estable (definición 3.3) y convergente (definición 3.1).

De la misma manera que el capítulo 2, esencialmente repetimos teoría matemática ya conocida a partir de diferentes fuente pero "digerida" a lo Séneca de forma tal de unificar nomenclaturas y criterios, en este hacemos lo mismo por cuestiones de consistencia. Mostramos algunos resultados conocidos y derivamos con algún cierto nivel de detalle razonable (teniendo en cuenta que es ésta una tesis de Ingeniería y no de Matemática) el problema de aproximación de Galerkin a partir de la formulación débil de un problema en derivadas parciales. Dejamos la derivación completa incluyendo la teoría de análisis funcional necesaria para demostrar completamente todos los resultados del método de elementos finitos en las referencias [12], [25], [44]. En la monografía [62] escrita durante el plan de formación de este doctorado se muestra una derivación de la formulación en elementos finitos de la ecuación de difusión multigrupo de forma menos formal pero más intuitiva. Incluso se comparan los resultados numéricos obtenidos con dicha formulación con los obtenidos con una formulación basada en volúmenes finitos [16].

Proposición 3.1. *Si se pudiera intercambiar en toda la literatura existente (y en las clases, seminarios, conferencias, etc.) la palabra "elementos" por "volúmenes" (¿tal vez con* `sed` *siguiendo la filosofía del Apéndice C?) nadie notaría la diferencia. Ver la referencia [35] y sus doscientas ochenta referencias para la historia detrás del "método de elementos finitos".*

Comenzamos ilustrando la aplicación el método de elementos finitos a un operador elíptico escalar, en particular a la ecuación de Poisson generalizada.[12] Para este caso introducimos las ideas básicas de

i. la formulación débil o variacional (sección 3.4.1),
ii. la aproximación de Galerkin (sección 3.4.1.4), y
iii. la discretización por elementos finitos (sección 3.4.1.5).

Luego en la sección 3.4.2 aplicamos estas ideas para obtener las versiones completamente discretizadas de las ecuaciones de difusión multigrupo, que también son elípticas pero el problema deja de ser un escalar en cada nodo espacial y su operador no es simétrico para $G > 1$. Finalmente en la sección 3.4.3 hacemos lo mismo para transporte por S_N multigrupo. En este caso la incógnita también tiene varios grados de libertad en cada nodo espacial. Pero además el operador es parabólico de primer orden y la formulación numérica requiere de un término de estabilización.

[12] En en apéndice B, más precisamente en la página 365, mostramos cómo aprovechar la elipticidad del operador de Laplace para resolver un laberinto arbitrario.

3.4.1. Ecuación de Poisson generalizada

Comencemos resolviendo la ecuación escalar elíptica de Poisson generalizada (en el sentido de que el coeficiente del operador diferencial puede depender del espacio) sobre un dominio espacial D-dimensional $U \in \mathbb{R}^D$—cuya frontera es ∂U—con condiciones de contorno de Dirichlet homogéneas en $\Gamma_D \in \partial U$ y condiciones arbitrarias de Neumann en $\Gamma_N \in \partial U$ tal que $\Gamma_D \cup \Gamma_N = \partial U$ y $\Gamma_D \cap \Gamma_N = \emptyset$ (figura 3.6):

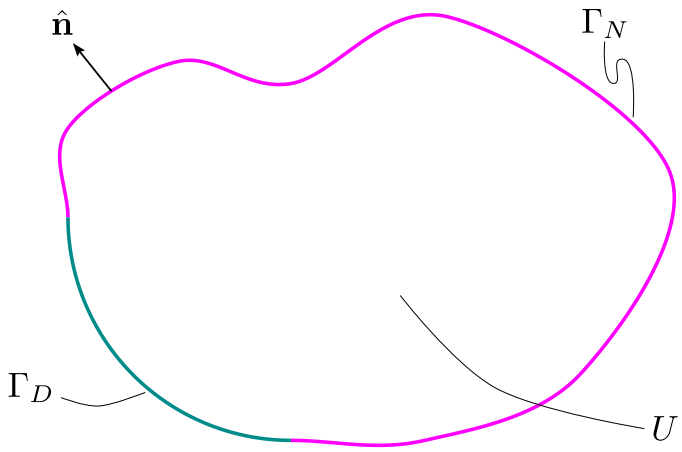


Figura 3.6.: El dominio espacial $U \in \mathbb{R}^2$ de la figura 1.6 con una parte de la frontera Γ_D con condiciones de Dirichlet (de color cian) y otra parte Γ_N con condiciones de Neumann (magenta).

$$\begin{cases} -\text{div}\Big[k(\mathbf{x}) \cdot \text{grad}\,[u(\mathbf{x})]\Big] = f(\mathbf{x}) & \forall \mathbf{x} \in U \\ u(\mathbf{x}) = 0 & \forall \mathbf{x} \in \Gamma_D \\ k(\mathbf{x}) \cdot \Big[\text{grad}\,[u(\mathbf{x})] \cdot \hat{\mathbf{n}}\Big] = p(\mathbf{x}) & \forall \mathbf{x} \in \Gamma_N \end{cases} \tag{3.22}$$

donde $\hat{\mathbf{n}}$ es la normal externa a la frontera ∂U en el punto $\mathbf{x}$.

3.4.1.1. Formulaciones fuertes y débiles

Definición 3.12 (formulación fuerte). Llamamos a la ecuación diferencial propiamente dicha junto con sus condiciones de contorno, tal como las escribimos en la ecuación 3.22, la *formulación fuerte* del problema.

Observación. En la formulación fuerte, todas las funciones deben ser derivables al menos hasta el orden apropiado según dónde aparezca cada una. Por ejemplo, en la ecuación 3.22, tanto u como el producto $k\nabla u$ deben ser derivables. Este requerimiento usualmente es demasiado restrictivo en aplicaciones físicas. Por ejemplo, la formulación fuerte del problema de conducción de calor no está bien definida en las interfaces entre materiales con diferentes conductividades k a cada lado de la interfaz.

Multipliquemos ambos miembros de la ecuación diferencial por una cierta función $v(\mathbf{x})$ que llamamos "de prueba":[13]

$$-v(\mathbf{x}) \cdot \operatorname{div}\Big[k(\mathbf{x}) \cdot \operatorname{grad}\left[u(\mathbf{x})\right]\Big] = v(\mathbf{x}) \cdot f(\mathbf{x}) \tag{3.23}$$

Esta función de prueba $v(\mathbf{x})$ puede ser (en principio) arbitraria, pero requerimos que se anule en Γ_D. Es decir, por ahora pedimos que $u(\mathbf{x})$ y $v(\mathbf{x})$ satisfagan las mismas condiciones de contorno de Dirichlet homogéneas (aunque no necesariamente las de Neumann).

Teorema 3.4 (de la divergencia). *En un dominio conexo $U \in \mathbb{R}^D$, la integral de volumen sobre U de la divergencia de una función vectorial continua $\mathbf{F}(\mathbf{x}) : U \mapsto \mathbb{R}^D$ es igual a la integral de superficie del producto interno entre $\mathbf{F}$ y la normal externa $\hat{\mathbf{n}}$ a la frontera ∂U:*

$$\int_U \operatorname{div}\left[\mathbf{F}(\mathbf{x})\right]\, d^D\mathbf{x} = \int_{\partial U} \mathbf{F}(\mathbf{x}) \cdot \hat{\mathbf{n}}\, d^{D-1}\mathbf{x}$$

Demostración. Cualquier libro de Análisis II.

□

Corolario 3.5 (fórmula de Green). *En un dominio conexo $U \in \mathbb{R}^D$, sean $u(\mathbf{x})$, $v(\mathbf{x})$ y $k(\mathbf{x})$ funciones continuas $U \mapsto \mathbb{R}$. Entonces*

$$\begin{aligned}\int_U v(\mathbf{x}) \cdot \operatorname{div}\Big[k(\mathbf{x}) \cdot \operatorname{grad}\left[u(\mathbf{x})\right]\Big]\, d^D\mathbf{x} = &- \int_U \operatorname{grad}\left[v(\mathbf{x})\right] \cdot k(\mathbf{x}) \cdot \operatorname{grad}\left[u(\mathbf{x})\right]\, d^D\mathbf{x} \\ &+ \int_{\partial U} v(\mathbf{x}) \cdot \Big[k(\mathbf{x}) \cdot \Big(\operatorname{grad}\left[u(\mathbf{x})\right] \cdot \hat{\mathbf{n}}\Big)\Big]\, d^{D-1}\mathbf{x}\end{aligned}$$

siendo $\hat{\mathbf{n}}$ la normal exterior a la frontera ∂U en el punto $\mathbf{x}$.

Demostración. Recordando el teorema 2.9 de la generalización de la derivada de un producto que dice que

$$\operatorname{div}[a \cdot \mathbf{b}] = a \cdot \operatorname{div}[\mathbf{b}] + \mathbf{b} \cdot \operatorname{grad}[a]$$

entonces para $a = v$ y $\mathbf{b} = k\nabla u$

$$\operatorname{div}\Big[v(\mathbf{x}) \cdot k(\mathbf{x}) \cdot \operatorname{grad}\left[u(\mathbf{x})\right]\Big] = v(\mathbf{x}) \cdot \operatorname{div}\Big[k(\mathbf{x}) \cdot \operatorname{grad}\left[u(\mathbf{x})\right]\Big] + k(\mathbf{x}) \cdot \operatorname{grad}\left[u(\mathbf{x})\right] \cdot \operatorname{grad}\left[v(\mathbf{x})\right]$$

Integrando sobre el volumen U[14]

[13] Del inglés *test funcion.*

[14] Llamamos volumen al dominio de dimensión D y superficie a la frontera de dimensión $D-1$.

$$\int_U \operatorname{div}\Big[v(\mathbf{x})\cdot k(\mathbf{x})\cdot \operatorname{grad}[u(\mathbf{x})]\Big]\, d^D\mathbf{x} = \int_U v(\mathbf{x})\cdot \operatorname{div}\Big[k(\mathbf{x})\cdot \operatorname{grad}[u(\mathbf{x})]\Big]\, d^D\mathbf{x} + \int_U k(\mathbf{x})\cdot \operatorname{grad}[u(\mathbf{x})]\cdot \operatorname{grad}[v(\mathbf{x})]\; d^D\mathbf{x}$$

Haciendo $\mathbf{F}(\mathbf{x}) = v(\mathbf{x})\cdot k(\mathbf{x})\cdot \operatorname{grad}[u(\mathbf{x})]$ en el teorema 3.4 tenemos

$$\int_U \operatorname{div}\Big[v(\mathbf{x})\cdot k(\mathbf{x})\cdot \operatorname{grad}[u(\mathbf{x})]\Big]\, d^D\mathbf{x} = \int_{\partial U} v(\mathbf{x})\cdot \Big[k(\mathbf{x})\cdot \Big(\operatorname{grad}[u(\mathbf{x})]\cdot \hat{\mathbf{n}}\Big)\Big]\; d^{D-1}\mathbf{x}$$

Igualando los miembros derechos de las últimas dos expresiones

$$\int_{\partial U} v(\mathbf{x})\cdot \Big[k(\mathbf{x})\cdot \Big(\operatorname{grad}[u(\mathbf{x})]\cdot \hat{\mathbf{n}}\Big)\Big]\; d^{D-1}\mathbf{x} = \int_U v(\mathbf{x})\cdot \operatorname{div}\Big[k(\mathbf{x})\cdot \operatorname{grad}[u(\mathbf{x})]\Big]\, d^D\mathbf{x} + \int_U k(\mathbf{x})\cdot \operatorname{grad}[u(\mathbf{x})]\cdot \operatorname{grad}[v(\mathbf{x})]\; d^D\mathbf{x}$$

Reordenando los términos, llegamos a la tesis del teorema.

□

Como $\Gamma_D \cup \Gamma_N = \partial U$ y $\Gamma_D \cap \Gamma_N = \emptyset$, entonces podemos escribir la integral de superficie sobre la frontera ∂U como suma de dos integrales con el mismo integrando, una sobre Γ_D y otra sobre Γ_N:

$$\int_{\partial U} v(\mathbf{x})\cdot \Big[k(\mathbf{x})\cdot \Big(\operatorname{grad}[u(\mathbf{x})]\cdot \hat{\mathbf{n}}\Big)\Big]\; d^{D-1}\mathbf{x} = \int_{\Gamma_D} v(\mathbf{x})\cdot \Big[k(\mathbf{x})\cdot \Big(\operatorname{grad}[u(\mathbf{x})]\cdot \hat{\mathbf{n}}\Big)\Big]\; d^{D-1}\mathbf{x} + \int_{\Gamma_N} v(\mathbf{x})\cdot \Big[k(\mathbf{x})\cdot \Big(\operatorname{grad}[u(\mathbf{x})]\cdot \hat{\mathbf{n}}\Big)\Big]\; d^{D-1}\mathbf{x}$$

Pero

i. habíamos pedido que $v(\mathbf{x})$ se anule en Γ_D

$$v(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \Gamma_D$$

ii. la condición de contorno de Neumann indica que

$$k(\mathbf{x})\cdot \Big[\operatorname{grad}[u(\mathbf{x})]\cdot \hat{\mathbf{n}}\Big] = p(\mathbf{x}) \quad \forall \mathbf{x} \in \Gamma_N$$

por lo que

$$\int_{\partial U} v(\mathbf{x}) \cdot \Big[k(\mathbf{x}) \cdot \Big(\operatorname{grad} [u(\mathbf{x})] \cdot \hat{\mathbf{n}} \Big) \Big] \; d^{D-1}\mathbf{x} = \int_{\Gamma_N} v(\mathbf{x}) \cdot p(\mathbf{x}) \, d^{D-1}\mathbf{x}$$

Volvamos a la ecuación 3.23 e integremos ambos miembros sobre el dominio U

$$- \int_U v(\mathbf{x}) \cdot \operatorname{div} \Big[k(\mathbf{x}) \cdot \operatorname{grad} [u(\mathbf{x})] \Big] \, d^D\mathbf{x} = \int_U v(\mathbf{x}) \cdot f(\mathbf{x}) \, d^D\mathbf{x}$$

Ahora usemos la fórmula de Green y el hecho de que $v(\mathbf{x})$ se anula en Γ_D para obtener

$$\int_U \operatorname{grad} [v(\mathbf{x})] \cdot k(\mathbf{x}) \cdot \operatorname{grad} [u(\mathbf{x})] \; d^D\mathbf{x} = \int_U v(\mathbf{x}) \cdot f(\mathbf{x}) \, d^D\mathbf{x} + \int_{\Gamma_N} p(\mathbf{x}) \cdot v(\mathbf{x}) \, d^{D-1}\mathbf{x} \tag{3.24}$$

Definición 3.13 (formulación débil). Llamamos a la expresión que resulta de

1. multiplicar ambos miembros de la ecuación diferencial por una función arbitraria llamada "de prueba" que se anula en Γ_D,
2. integrar sobre el dominio espacial,
3. aplicar fórmulas de cálculo vectorial, y
4. reemplazar la condición de contorno de Neumann en las integrales de superficie

tal como la ecuación 3.24, junto con los requerimientos que deben satisfacer tanto la función incógnita como la función de prueba, la *formulación débil* o *variacional* del problema. Estrictamente hablando, la formulación débil de una ecuación diferencial es

$$\text{encontrar } u(\mathbf{x}) \in V : \quad a\Big(u(\mathbf{x}), v(\mathbf{x})\Big) = \mathcal{B}\Big(v(\mathbf{x})\Big) \quad \forall v(\mathbf{x}) \in V$$

donde V es un espacio funcional apropiado, por ejemplo el $H_0^1(U)$ de las funciones $U \in \mathbb{R}^D \mapsto \mathbb{R}$ cuyo gradiente (el superíndice uno) es de cuadrado integrable en el dominio U y que se anulan en Γ_D (el subíndice cero)

$$V = H_0^1(U) = \left\{ v \in H_0^1(U) : \int_U (\nabla v)^2 \; d^D\mathbf{x} < \infty \wedge v(\mathbf{x}) = 0 \forall \mathbf{x} \in \Gamma_D \right\}$$

y los operadores $a(u, v) : V \times V \mapsto \mathbb{R}$ y $\mathcal{B}(v) : V \mapsto \mathbb{R}$ se obtienen a partir de los cuatro pasos arriba mencionados. En particular, para el problema generalizado de Poisson de la formulación de la ecuación 3.24, es

$$\begin{aligned} a(u, v) &= \int_U \operatorname{grad}\Big[v(\mathbf{x})\Big] \cdot k(\mathbf{x}) \cdot \operatorname{grad}\Big[u(\mathbf{x})\Big] \, d^D\mathbf{x} \\ \mathcal{B}(v) &= \int_U v(\mathbf{x}) \cdot f(\mathbf{x}) \, d^D\mathbf{x} + \int_{\Gamma_N} v(\mathbf{x}) \cdot p(\mathbf{x}) \, d^{D-1}\mathbf{x} \end{aligned} \tag{3.25}$$

Observación. En la formulación débil la derivabilidad es más laxa que en la formulación fuerte. De ahí su nombre: las funciones deben cumplir requerimientos más débiles. Por un lado, al involucrar una operación de integración sobre el dominio y aplicar fórmulas de Green, los requerimientos de derivabilidad disminuyen un grado: en la formulación fuerte 3.22, u tiene que ser derivable dos veces ya que el operador es esencialmente el laplaciano mientras que en la formulación débil 3.24 sólo involucra el gradiente. De hecho, ni siquiera hace falta que las funciones sean tan derivables según en el lugar dónde aparecen en la formulación ya que las las integrales deben tomarse según el sentido de Lebesgue y no según el sentido de como Riemann: todas las funciones dentro de las integrales pueden ser discontinuas en un sub-espacio de medida nula. En efecto, la formulación débil del problema de conducción de calor con conductividad discontinua en interfaces materiales está bien definida. Por un lado las interfaces materiales son un sub-espacio de medida nula y por otro la conductividad $k(\mathbf{x})$ no tiene aplicado ningún operador diferencial sino que es integrado (en el sentido de Lebesgue) sobre el dominio espacial U.

Observación. La formulación débil de la ecuación de conducción de calor derivada en la ecuación 3.24 incluye la posibilidad de que la conductividad $k(\mathbf{x})$ pueda depender del espacio e incluso ser discontinua en interfaces materiales. Más aún, la derivación propuesta puede ser extendida para el caso no lineal en el cual la conductividad pueda depender de la incógnita $k(u)$. Ver por ejemplo el problema de conducción de calor no lineal del sección B.3.1.3.

Observación. El nombre *variacional* viene del hecho de requerir que $a(u, v) = \mathcal{B}(v)$ para todas las posibles funciones de prueba $v(\mathbf{x}) \in V$. Es decir, de requerir que v pueda "variar" arbitrariamente (siempre que se anule en Γ_D) y la igualdad se siga manteniendo.

Observación. La formulación fuerte incluye las condiciones de contorno en su enunciado. Las condiciones de Neumann aparecen naturalmente en los términos de superficie luego de aplicar las fórmulas de Green y las condiciones de Dirichlet están esencialmente en el espacio vectorial V donde se busca la solución u. Las primeras se llaman *naturales* y las segundas *esenciales*.

Teorema 3.5. *El problema débil es equivalente al fuerte en el sentido de distribuciones, es decir, ambas formulaciones coinciden excepto en a lo más un sub-conjunto de U de medida cero.*

Demostración. Sección 3.3.2 de [44], teorema 0.1.4 de [12] y/o sección 1.4 de [25].

□

Definición 3.14 (funcional lineal). Un funcional $\mathcal{B}(v) : V \mapsto \mathbb{R}$ es lineal si

$$\mathcal{B}(\alpha \cdot v_1 + \beta \cdot v_2) = \alpha \cdot \mathcal{B}(v_1) + \beta \cdot \mathcal{B}(v_2)$$

Definición 3.15 (operador bi-lineal). Un operador $a(v, u) : V \times V \mapsto \mathbb{R}$ es bi-lineal si

$$a(\alpha \cdot v_1 + \beta \cdot v_2, u) = \alpha \cdot a(v_1, u) + \beta \cdot a(v_2, u)$$

y

$$a(v, \alpha \cdot u_1 + \beta \cdot u_2) = \alpha \cdot a(v, u_1) + \beta \cdot a(v, u_2)$$

Definición 3.16 (operador simétrico). Un operador $a(v,u)$ es simétrico si

$$a(v,u) = a(u,v)$$

Definición 3.17 (operador coercitivo). Un operador $a(v,u) : V \times V \mapsto \mathbb{R}$ es coercitivo si existe una constante $\alpha > 0$ tal que

$$a(v,v) \geq \alpha \cdot ||v||_V^2$$

Corolario 3.6. *Si $a(v,u)$ es coercitivo entonces*

$$||v||_a = \sqrt{a(v,v)}$$

es una norma.

Teorema 3.6. *El operador*

$$a(u,v) = \int_U \text{grad}\Big[v(\mathbf{x})\Big] \cdot k(\mathbf{x}) \cdot \text{grad}\Big[u(\mathbf{x})\Big]\, d^D\mathbf{x}$$

es coercitivo si $k(\mathbf{x}) > 0 \forall \mathbf{x} \in U$.

Demostración. La demostración detallada se puede encontrar la sección 5.3 de [12] e en involucra análisis funcional y algunas desigualdades, como la de Poincaré. La idea básica es que $\int_U [\nabla v]^2 d^D\mathbf{x}$ se comporta en forma similar a $\int_U v^2 d^D\mathbf{x}$.

□

Teorema 3.7 (de Lax-Milgram). *Dada una formulación débil*

$$\textit{encontrar } u \in V : \quad a(u,v) = \mathcal{B}(v) \quad \forall v \in V$$

siendo

a. V un sub-espacio de $H^1(U)$,
b. $a : V \times V \mapsto \mathbb{R}$ un operador continuo, bi-lineal y coercitivo, y
c. $\mathcal{B} : V \mapsto \mathbb{R}$ un funcional continuo y lineal

entonces la solución u existe y es única.

Demostración. Sección 2.7 de [12] o sección 3.5 de [44].

□

3.4.1.2. Condiciones de contorno de Dirichlet no homogéneas

Hasta ahora las condiciones de contorno de Dirichlet han sido iguales a cero, ya que al pedir que tanto la incógnita u como las funciones de prueba v pertenezcan a H_0^1 podemos

1. satisfacer las condiciones esenciales sobre u, y
2. anular el término de superficie sobre Γ_D de la fórmula de Green

Si el problema a resolver tiene una condición de contorno no homogénea, digamos

$$\begin{cases} -\text{div}\Big[k(\mathbf{x}) \cdot \text{grad}\left[u(\mathbf{x})\right]\Big] = f(\mathbf{x}) & \forall \mathbf{x} \in U \\ u(\mathbf{x}) = g(\mathbf{x}) & \forall \mathbf{x} \in \Gamma_D \\ k(\mathbf{x}) \cdot \Big[\text{grad}\left[u(\mathbf{x})\right] \cdot \hat{\mathbf{n}}\Big] = p(\mathbf{x}) & \forall \mathbf{x} \in \Gamma_N \end{cases} \tag{3.26}$$

entonces una idea sería pedir que $v \in H_0^1$ pero que $u \in H_g^1$ tal que

$$H_g^1(U) = \left\{ v \in H_g^1(U) : \int_U (\nabla v)^2 \, d^D\mathbf{x} < \infty \wedge v(\mathbf{x}) = g(\mathbf{x}) \; \forall \mathbf{x} \in \Gamma_D \right\}$$

Este planteo, además de ser poco elegante al romper la simetría entre u y v, tiene un problema insalvable: H_g^1 es un conjunto[15] pero no un espacio ya que la suma de dos funciones $u_1 \in H_g^1$ y $u_2 \in H_g^1$ no están en H_g^1 sino en H_{2g}^1. Esto hace que no podamos escribir fácilmente a la incógnita u como una combinación lineal de una base, que es lo primero que hacemos en la sección 3.4.1.4 que sigue.

Una alternativa es considerar una función continua $u_g \in H_g^1$ y escribir

$$u_h(\mathbf{x}) = u(\mathbf{x}) - u_g(\mathbf{x}) \tag{3.27}$$

donde $u_h \in H_0^1$, es decir, se anula en Γ_D (el subíndice h quiere decir "homogénea"). Si el operador a es bi-lineal, entonces podemos escribir el problema

$$\text{encontrar } u \in V : \quad a(u, v) = \mathcal{B}(v) \quad \forall v \in V$$

como

$$\text{encontrar } u_h \in V : \quad a(u_h, v) = \mathcal{B}(v) - a\left(u_g, v\right) \quad \forall v \in V$$

donde ahora tanto la incógnita parcial u_h como las funciones de prueba v pertenecen a $V = H_0^1$ y todos los datos del problema están en el miembro derecho de la igualdad. Podemos obtener la incógnita original u a partir de la ecuación 3.27 como

$$u(\mathbf{x}) = u_h(\mathbf{x}) + u_g(\mathbf{x})$$

[15] Técnicamente es un *affine manifold*.

Observación. Si bien este procedimiento es matemáticamente correcto, no parece sencillo encontrar una función $u_g \in H^1_g$ apropiada para una condición de contorno arbitraria $g(\mathbf{x})$. En la sección 3.4.1.5 mostramos que en un espacio vectorial de dimensión finita el procedimiento es más sencillo. En el capítulo 4 que sigue mostramos que la implementación práctica de este tipo de condiciones de contorno es más sencilla todavía.

3.4.1.3. Condiciones de contorno de Robin

Si el problema tiene una condición de contorno de Robin, digamos

$$\gamma(\mathbf{x}) \cdot u(\mathbf{x}) + k(\mathbf{x}) \cdot \Big[\operatorname{grad}[u(\mathbf{x})] \cdot \hat{\mathbf{n}}\Big] = \beta(\mathbf{x}) \quad \forall \mathbf{x} \in \Gamma_R$$

podemos pasar el primer término al otro miembro y llamar $p(\mathbf{x})$ a la expresión resultante

$$k(\mathbf{x}) \cdot \Big[\operatorname{grad}[u(\mathbf{x})] \cdot \hat{\mathbf{n}}\Big] = \beta(\mathbf{x}) - \gamma(\mathbf{x}) \cdot u(\mathbf{x}) = p(\mathbf{x})$$

De esta manera, una condición de Robin tendrá una contribución sobre el operador $a(u, v)$

$$\int_{\Gamma_R} v(\mathbf{x}) \cdot \gamma(\mathbf{x}) \cdot u(\mathbf{x})\, d^{D-1}\mathbf{x}$$

y otra contribución sobre el funcional $\mathcal{B}(v)$

$$\int_{\Gamma_R} u(\mathbf{x}) \cdot \beta(\mathbf{x})\, d^{D-1}\mathbf{x}$$

Observación. Si $\gamma(\mathbf{x})$ es lo suficientemente negativo, el operador $a(u, v)$ puede perder su coercitividad.

3.4.1.4. Aproximación de Galerkin

Usando las ideas desarrolladas en la sección anterior, podemos definir una discretización espacial muy fácilmente como sigue.

Definición 3.18 (problema de Galerkin). Sea V_N un sub-espacio de $V = H^1_0(U)$ de dimensión finita N. Llamamos *problema de Galerkin* a

$$\text{encontrar } u_N \in V_N : \quad a(u_N, v_N) = \mathcal{B}(v_N) \quad \forall v_N \in V_N$$

Como V_N es un espacio vectorial de dimensión N entonces podemos encontrar N funciones $h_i(\mathbf{x})$ que formen una base de V_N

$$V_N = \text{span}\Big\{h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_N(\mathbf{x})\Big\}$$

En efecto, $v_N \in V_N$ puede ser escrita como una combinación lineal de las N funciones $h_i(\mathbf{x})$

$$v_N(\mathbf{x}) = \sum_{i=1}^{N} v_i \cdot h_i(\mathbf{x}) \tag{3.28}$$

Para a bi-lineal y $\mathcal{B}$ lineal,

$$\begin{aligned}
0 &= a\Big(u_N(\mathbf{x}), v_N(\mathbf{x})\Big) - \mathcal{B}\Big(v_N(\mathbf{x})\Big) \\
0 &= a\left(u_N(\mathbf{x}), \sum_{i=1}^{N} v_i \cdot h_i\left(\mathbf{x}\right)\right) - \mathcal{B}\left(\sum_{i=1}^{N} v_i \cdot h_i\left(\mathbf{x}\right)\right) \\
0 &= \sum_{i=1}^{N} v_i \cdot \Big[a\Big(u_N(\mathbf{x}), h_i\left(\mathbf{x}\right)\Big) - \mathcal{B}\Big(h_i\left(\mathbf{x}\right)\Big)\Big]
\end{aligned}$$

Como esta igualdad tiene que cumplirse $\forall v_N \in V_N$ entonces cada corchete debe anularse independientemente de v_i, lo que implica que

$$a\Big(u_N(\mathbf{x}), h_i\left(\mathbf{x}\right)\Big) = \mathcal{B}\Big(h_i\left(\mathbf{x}\right)\Big) \quad \text{para } i = 1, \dots, N$$

De la misma manera, $u_N \in V_N$ por lo que $u_N(\mathbf{x}) = \sum_{j=1}^{N} u_j \cdot h_j(\mathbf{x})$ entonces

$$\begin{aligned}
a\Big(\sum_{j=1}^{N} u_j \cdot h_j(\mathbf{x}), h_i\left(\mathbf{x}\right)\Big) &= \mathcal{B}\Big(h_i\left(\mathbf{x}\right)\Big) \\
\sum_{j=1}^{N} a\Big(h_i(\mathbf{x}), h_j\left(\mathbf{x}\right)\Big) \cdot u_j &= \mathcal{B}\Big(h_i\left(\mathbf{x}\right)\Big) \quad \text{para } i = 1, \dots, N
\end{aligned}$$

que podemos escribir en forma matricial como

$$\mathsf{A} \cdot \mathbf{u} = \mathbf{b} \tag{3.29}$$

siendo

$$\mathsf{A} = \begin{bmatrix}
a(h_1,h_1) & a(h_1,h_2) & \cdots & a(h_1,h_j) & \cdots & a(h_1,h_N) \\
a(h_2,h_1) & a(h_2,h_2) & \cdots & a(h_2,h_j) & \cdots & a(h_2,h_N) \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
a(h_i,h_1) & a(h_i,h_2) & \cdots & a(h_i,h_j) & \cdots & a(h_i,h_N) \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
a(h_N,h_1) & a(h_N,h_2) & \cdots & a(h_N,h_j) & \cdots & a(h_N,h_N)
\end{bmatrix}$$

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_i \\ \vdots \\ u_N \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} \mathcal{B}(h_1) \\ \mathcal{B}(h_2) \\ \vdots \\ \mathcal{B}(h_i) \\ \vdots \\ \mathcal{B}(h_N) \end{bmatrix}$$

Corolario 3.7. *Si el operador a es simétrico entonces la matriz A también es simétrica.*

Teorema 3.8. *Si el operador a es bi-lineal y coercitivo entonces la matriz A es definida positiva.*

Demostración.

$$\begin{aligned} \mathbf{v}^T \cdot \mathsf{A} \cdot \mathbf{v} &= \sum_{i=1}^{N} \sum_{j=1}^{N} v_i \cdot a(h_i, h_j) \cdot v_j \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} a(v_i \cdot h_i, v_j \cdot h_j) \\ &= a\left(\sum_{i=1}^{N} v_i \cdot h_i, \sum_{j=1}^{N} v_j \cdot h_j \right) \\ &= a(v_N, v_N) \geq \alpha \cdot ||v_N||_V^2 \geq 0 \end{aligned}$$

para $\alpha > 0$. Dado que $||v_N||_V$ es una norma, la igualdad se cumple si y sólo si $||v_N||_V = 0$, lo que implica que todos los elementos de v son nulos.

□

Observación. Ver [25] para una demostración alternativa.

Teorema 3.9 (existencia y unicidad). *Si el operador a es bi-lineal y coercitivo entonces el problema de Galerkin de la definición 3.18 existe y es único.*

Demostración. Por el teorema 3.8 la matriz A es definida positiva. Luego es invertible y la ecuación 3.29 $\mathsf{A} \cdot \mathbf{u} = \mathbf{b}$ tiene solución única.

□

Teorema 3.10 (estabilidad). *El método de Galerkin es estable con respecto a N.*

Demostración. Sección 4.2.2 de [44].

□

Teorema 3.11 (consistencia). *El método de Galerkin es fuertemente consistente, es decir*

$$a(u - u_N, v_N) = 0 \quad \forall v_N \in V_N$$

Demostración. Como u_N es una solución del problema de Galerkin entonces

$$a(u_N, v_N) = \mathcal{B}(v_N) \quad \forall v_N \in V_N$$

Dado que $V_N \subset V$ entonces la solución continua $u \in V$ también satisface

$$a(u, v_N) = \mathcal{B}(v_N) \quad \forall v_N \in V_N$$

Restando miembro a miembro

$$a(u_N, v_N) - a(u, v_N) = 0$$

de donde se sigue la tesis por la bi-linealidad.

□

Observación. El error $u - u_N$ cometido por la aproximación de Galerkin es ortogonal al sub-espacio V_N en la norma

$$||v||_a = \sqrt{a(v, v)}$$

Es decir, la solución aproximada u_N es

1. La proyección ortogonal de la solución exacta u en el sub-espacio V_N.
2. La solución que minimiza la distancia $||u - u_N||_a$.

Corolario 3.8 (convergencia). *Si $V_N \to V$ para $N \to \infty$ entonces el método de Galerkin converge a la solución real u.*

Observación. En esta sección 3.4.1 se ha comenzado con la formulación fuerte de la ecuación diferencial (ecuación 3.22) y se ha llegado a un sistema lineal de ecuaciones algebraica (ecuación 3.29), pasando por la formulación débil (definición 3.13) y por la aproximación de Galerkin (definición 3.18):

$$\text{formulación fuerte} \quad \equiv \quad \text{formulación débil} \quad \approx\!\!\approx \quad \text{aproximación de Galerkin} \equiv \quad \mathsf{A} \cdot \mathbf{u} = \mathbf{b}$$

La primera equivalencia está probada por el teorema 3.5. No hay ninguna aproximación involucrada. Solamente hay que marcar que la equivalencia se mantiene en todo el dominio U excepto en, a lo más, un sub-conjunto de medida cero. La aproximación entre la formulación débil y el problema de Galerkin es la idea central del método numérico: pasar de una espacio vectorial V de dimensión infinita a un espacio vectorial V_N de dimensión finita. La equivalencia entre Galerkin y un sistema lineal de ecuaciones algebraicas (que puede ser resuelto con una computadora digital) funciona siempre y cuando el operador $a(u, v)$ sea coercitivo y bi-lineal. Para problemas no lineales (por ejemplo para el caso en el que k dependiera de u) la última equivalencia se reemplaza por una formulación vectorial no lineal $\mathbf{F}(\mathbf{u}) = 0$. En la sección 3.5.1 mencionamos brevemente cómo formular y resolver este tipo de problemas.

3.4.1.5. Elementos finitos

Tomemos un dominio $U \in \mathbb{R}^D$ y consideremos J puntos $\mathbf{x}_j \in U$. Estos puntos $\mathbf{x}_j$ para $j = 1, \dots, J$ incluyen la frontera Γ_N con condiciones de contorno de Neumann pero no incluyen a Γ_D con condiciones de Dirichlet. Por ejemplo, en la figura 3.7 tenemos $J = 32$. Supongamos que existen J funciones $h_j(\mathbf{x})$ "de forma"[16] que cumplen simultáneamente

$$\begin{cases} h_j(\mathbf{x}_i) = \delta_{ji} \\ h_j(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \Gamma_D \end{cases} \tag{3.30}$$

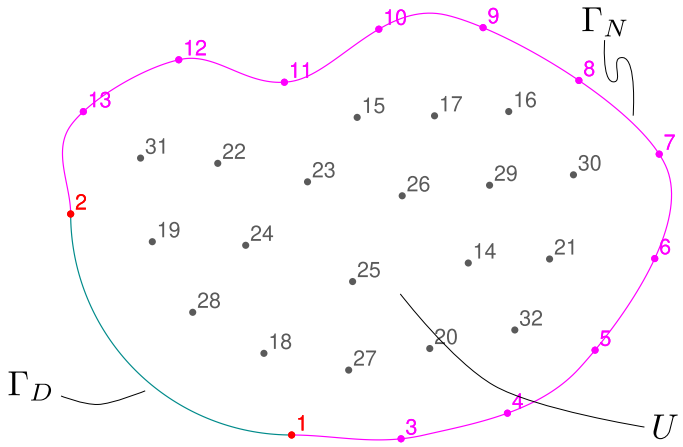


Figura 3.7.: El dominio espacial U de la figura 1.6 con $J = 32$ puntos ubicados en la frontera Γ_N ($j = 1, \dots, 13$) y en el interior ($j = 14, \dots, 32$). La frontera Γ_D con condiciones de contorno de Dirichlet no tiene ningún punto.

Sea V_J el espacio vectorial de dimensión J generado por estas J funciones de forma $h_j(\mathbf{x})$. Como ya hicimos en la ecuación 3.28, escribimos a una cierta función $v(\mathbf{x}) \in V_J$ como una combinación lineal de los elementos de la base

$$v(\mathbf{x}) = \sum_{j=1}^{J} v_j \cdot h_j(\mathbf{x}) \tag{3.31}$$

Podemos escribir esta expansión en forma matricial como

$$v(\mathbf{x}) = \mathsf{H}(\mathbf{x}) \cdot \mathbf{v} = \mathbf{v}^T \cdot \mathsf{H}^T(\mathbf{x})$$

[16] En la gran mayoría de la literatura de elementos finitos las funciones de forma se llaman $N(\mathbf{x})$. Como este símbolo no nos parece apropiado para una función del espacio, seguimos la nomenclatura de Bathe [9] (que fue director de doctorado del Dr. Dvorkin que a su vez organizó el departamento de cálculo de la UBA donde este doctorando cursó la materia de elementos finitos) que utiliza la nomenclatura $h(\mathbf{x})$ para las funciones de forma.

con[17]

$$\mathsf{H}(\mathbf{x}) = \begin{bmatrix} h_1(\mathbf{x}) & h_2(\mathbf{x}) & \cdots & h_j(\mathbf{x}) & \cdots & h_J(\mathbf{x}) \end{bmatrix} \tag{3.32}$$

y

$$\mathbf{v} = \begin{bmatrix} v(\mathbf{x}_1) \\ v(\mathbf{x}_2) \\ \vdots \\ (\mathbf{x}_j) \\ \vdots \\ (\mathbf{x}_J) \end{bmatrix}$$

De la misma forma, el gradiente ∇v es

$$\text{grad}\Big[v(\mathbf{x})\Big] = \begin{bmatrix} \dfrac{\partial v}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial v}{\partial z} \end{bmatrix} = \begin{bmatrix} \displaystyle\sum_{j=1}^{J} v^{(j)} \cdot \frac{\partial h_j}{\partial x} \\ \displaystyle\sum_{j=1}^{J} v^{(j)} \cdot \frac{\partial h_j}{\partial y} \\ \displaystyle\sum_{j=1}^{J} v^{(j)} \cdot \frac{\partial h_j}{\partial z} \end{bmatrix} = \mathsf{B}(\mathbf{x}) \cdot \mathbf{v}$$

con

$$\mathsf{B}(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial h_1}{\partial x} & \dfrac{\partial h_2}{\partial x} & \cdots & \dfrac{\partial h_j}{\partial x} & \cdots & \dfrac{\partial h_J}{\partial x} \\ \dfrac{\partial h_1}{\partial y} & \dfrac{\partial h_2}{\partial y} & \cdots & \dfrac{\partial h_j}{\partial y} & \cdots & \dfrac{\partial h_J}{\partial y} \\ \dfrac{\partial h_1}{\partial z} & \dfrac{\partial h_2}{\partial z} & \cdots & \dfrac{\partial h_j}{\partial z} & \cdots & \dfrac{\partial h_J}{\partial z} \end{bmatrix} \tag{3.33}$$

Reemplazando la forma particular del operador a y del funcional $\mathcal{B}$ para el problema generalizado de Poisson de la ecuación 3.25, tenemos

$$\begin{aligned} a(u,v) &= \int_U \text{grad}\Big[v(\mathbf{x})\Big] \cdot k(\mathbf{x}) \cdot \text{grad}\Big[u(\mathbf{x})\Big]\, d^D\mathbf{x} \\ &= \int_U \mathbf{v}^T \cdot \mathsf{B}^T(\mathbf{x}) \cdot k(\mathbf{x}) \cdot \mathsf{B}(\mathbf{x}) \cdot \mathbf{u}\; d^D\mathbf{x} \\ &= \mathbf{v}^T \cdot \left[\int_U \mathsf{B}^T(\mathbf{x}) \cdot k(\mathbf{x}) \cdot \mathsf{B}(\mathbf{x})\, d^D\mathbf{x}\right] \cdot \mathbf{u} \end{aligned}$$

[17] Si bien la nomenclatura usual es llamar v_j a los componentes del vector $\mathbf{v}$, dejamos el subíndice para indicar grupos de energía al discretizar las ecuaciones de neutrónica en las secciones que siguen.

$$\begin{aligned}\mathcal{B}(v) &= \int_U v(\mathbf{x}) \cdot f(\mathbf{x})\, d^D\mathbf{x} + \int_{\Gamma_N} v(\mathbf{x}) \cdot p(\mathbf{x})\, d^{D-1}\mathbf{x} \\ &= \int_U \mathbf{v}^T \cdot \mathsf{H}^T(\mathbf{x}) \cdot f(\mathbf{x})\, d^D\mathbf{x} + \int_{\Gamma_N} \mathbf{v}^T \cdot \mathsf{H}^T(\mathbf{x}) \cdot p(\mathbf{x})\, d^{D-1}\mathbf{x} \\ &= \mathbf{v}^T \cdot \left[\int_U \mathsf{H}^T(\mathbf{x}) \cdot f(\mathbf{x})\, d^D\mathbf{x} + \int_{\Gamma_N} \mathsf{H}^T(\mathbf{x}) \cdot p(\mathbf{x})\, d^{D-1}\mathbf{x} \right]\end{aligned}$$

Como $a(u,v) = \mathcal{B}(v) \quad \forall v \in V_J$ entonces llegamos otra vez a

$$\mathsf{A} \cdot \mathbf{u} = \mathbf{b}$$

donde ahora tenemos una representación explícita particular para $\mathsf{A} \in \mathbb{R}^{J\times J}$ y $\mathbf{u} \in \mathbb{R}^J$ a partir de las ecuaciones [-ecuación 3.32} y 3.33 como

$$\begin{aligned}\mathsf{A} &= \int_U \mathsf{B}^T(\mathbf{x}) \cdot k(\mathbf{x}) \cdot \mathsf{B}(\mathbf{x})\, d^D\mathbf{x} \\ \mathbf{b} &= \int_{\Gamma_N} \mathsf{H}^T(\mathbf{x}) \cdot p(\mathbf{x})\, d^{D-1}\mathbf{x} + \int_{\Gamma_N} \mathsf{H}^T(\mathbf{x}) \cdot p(\mathbf{x})\, d^{D-1}\mathbf{x}\end{aligned} \tag{3.34}$$

Una vez más, tal como hemos dicho en la observación sobre la construcción de la función u_g necesaria para satisfacer condiciones de contorno de Dirichlet no homogéneas de la página 114, estas últimas dos expresiones son correctas. Pero no parece sencillo…

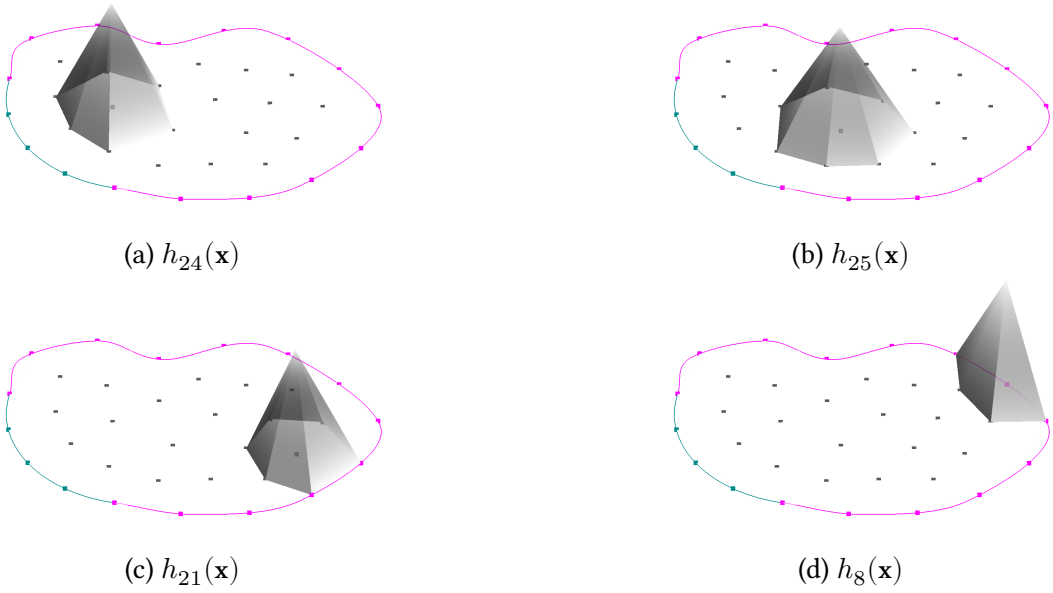
1. encontrar las J funciones de forma $h_j(\mathbf{x})$ adecuadas que cumplan las condiciones 3.30 (como por ejemplo las ilustradas en la figura 3.8), ni
2. calcular las integrales para obtener la matriz $\mathsf{A} \in \mathbb{R}^{J\times J}$ y el vector $\mathbf{b} \in \mathbb{R}^J$.

Justamente, el método de elementos finitos propone una forma sistemática para atacar estos dos puntos a partir de explotar la topología de los J puntos $\mathbf{x}_j$ de la figura 3.7. El hecho de no haber incluido puntos sobre la frontera Γ_D en el conjunto de J funciones de forma de alguna manera rompe el sistematismo necesario para aplicar el método. Lo primero que tenemos que hacer entonces es incluir puntos sobre la frontera Γ_D. Digamos que hay J_D puntos sobre Γ_D. Entonces agregamos J_D funciones de forma para $j = J+1, \dots, J+J_D$ a las cuales les pedimos que

$$h_j(\mathbf{x}_i) = \delta_{ji} \quad \text{para} \quad j = J+1, \dots, J+J_D \quad \text{e} \quad i = 1, \dots, J+J_D$$

Es decir, que estas nuevas funciones de forma se anulen en los demás $J+J_D-1$ puntos pero no necesitamos que se anulen en la frontera como le pedíamos a las primeras funciones de forma "originales" para $j \leq J$. Como las funciones de forma originales cumplen las condiciones de la ecuación 3.30, es decir sí se anulan en

a. todos los nodos diferentes de j, y
b. en todos los puntos $\mathbf{x} \in \Gamma_D$



(a) $h_{24}(\mathbf{x})$ (b) $h_{25}(\mathbf{x})$

(c) $h_{21}(\mathbf{x})$ (d) $h_8(\mathbf{x})$

Figura 3.8.: Funciones de forma de primer orden apropiadas para diferentes puntos de la figura 3.7. Una de las preguntas centrales que el método de elementos finitos responde es ¿cómo encontrarlas?

entonces también cumplen

$$h_j(\mathbf{x}_i) = \delta_{ji} \quad \text{para} \quad j = 1, \dots, J \quad \text{e} \quad i = 1, \dots, J + J_D$$

Luego

$$h_j(\mathbf{x}_i) = \delta_{ji} \quad \text{para} \quad j = 1, \dots, J + J_D \quad \text{e} \quad i = 1, \dots, J + J_D$$

y recuperamos una parte la sistematicidad requerida para aplicar el método de elementos finitos. Para recuperar la otra parte re-escribimos la ecuación 3.31 poniendo coeficientes iguales a cero en las funciones de forma sobre Γ_D

$$v(\mathbf{x}) = \sum_{j=1}^{J} v^{(j)} \cdot h_j(\mathbf{x}) = \sum_{j=1}^{J} v^{(j)} \cdot h_j(\mathbf{x}) + \sum_{j=J+1}^{J+J_D} 0 \cdot h_j(\mathbf{x})$$

que, en forma matricial, queda

$$v(\mathbf{x}) = \tilde{\mathsf{H}}(\mathbf{x}) \cdot \tilde{\mathbf{v}} = \tilde{\mathbf{v}}^T \cdot \tilde{\mathsf{H}}^T(\mathbf{x})$$

donde ahora los objetos tildados son objetos "extendidos" incluyendo los J_D puntos sobre Γ_D como

$$\tilde{\mathsf{H}}(\mathbf{x}) = \begin{bmatrix} h_1(\mathbf{x}) & h_2(\mathbf{x}) & \cdots & h_j(\mathbf{x}) & \cdots & h_J(\mathbf{x}) & h_{J+1}(\mathbf{x}) & \cdots & h_{J+J_D}(\mathbf{x}) \end{bmatrix} \tag{3.35}$$

y

$$\tilde{\mathbf{v}} = \begin{bmatrix} v^{(1)} \\ v^{(2)} \\ \vdots \\ v^{(j)} \\ \vdots \\ v^{(J)} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

De la misma manera extendemos la matriz $\mathsf{B}(\mathbf{x})$ como

$$\tilde{\mathsf{B}}(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial h_1}{\partial x} & \dfrac{\partial h_2}{\partial x} & \cdots & \dfrac{\partial h_j}{\partial x} & \cdots & \dfrac{\partial h_J}{\partial x} & \dfrac{\partial h_{J+1}}{\partial x} & \cdots & \dfrac{\partial h_{J+J_D}}{\partial x} \\ \dfrac{\partial h_1}{\partial y} & \dfrac{\partial h_2}{\partial y} & \cdots & \dfrac{\partial h_j}{\partial y} & \cdots & \dfrac{\partial h_J}{\partial y} & \dfrac{\partial h_{J+1}}{\partial y} & \cdots & \dfrac{\partial h_{J+J_D}}{\partial y} \\ \dfrac{\partial h_1}{\partial z} & \dfrac{\partial h_2}{\partial z} & \cdots & \dfrac{\partial h_j}{\partial z} & \cdots & \dfrac{\partial h_J}{\partial z} & \dfrac{\partial h_{J+1}}{\partial z} & \cdots & \dfrac{\partial h_{J+J_D}}{\partial z} \end{bmatrix} \tag{3.36}$$

Repitiendo todos los pasos, el método de Galerkin requiere que

$$\tilde{\mathbf{v}}^T \cdot \left[\int_U \tilde{\mathsf{B}}^T(\mathbf{x}) \cdot k(\mathbf{x}) \cdot \tilde{\mathsf{B}}(\mathbf{x}) \, d^D\mathbf{x} \right] \cdot \tilde{\mathbf{u}} = \tilde{\mathbf{v}}^T \cdot \left[\int_U \tilde{\mathsf{H}}^T(\mathbf{x}) \cdot f(\mathbf{x}) \, d^D\mathbf{x} + \int_{\Gamma_N} \tilde{\mathsf{H}}^T(\mathbf{x}) \cdot p(\mathbf{x}) \, d^{D-1}\mathbf{x} \right] \tag{3.37}$$

para todo $\tilde{v}^T = \begin{bmatrix} v^{(j)} & \cdots & v^{(J)} & 0 & \cdots & 0 \end{bmatrix}$.

Teorema 3.12. *Este requerimiento es equivalente a* $\mathsf{A} \cdot \mathbf{u} = \mathbf{b}$.

Demostración. Sean los objetos extendidos

$$\tilde{\mathbf{v}} = \begin{bmatrix} \mathbf{v} \\ \mathbf{0} \end{bmatrix} \quad \tilde{\mathsf{A}} = \begin{bmatrix} \mathsf{A} & \mathsf{C} \\ \mathsf{D} & \mathsf{E} \end{bmatrix} \quad \tilde{\mathbf{u}} = \begin{bmatrix} \mathbf{u} \\ \mathbf{0} \end{bmatrix} \quad \tilde{\mathbf{b}} = \begin{bmatrix} \mathbf{b} \\ \mathbf{e} \end{bmatrix}$$

Entonces

$$\begin{aligned} \tilde{\mathbf{v}}^T \cdot \tilde{\mathsf{A}} \cdot \tilde{\mathbf{u}} &= \tilde{\mathbf{v}}^T \cdot \tilde{\mathbf{b}} \\ \begin{bmatrix} \mathbf{v}^T & \mathbf{0}^T \end{bmatrix} \cdot \begin{bmatrix} \mathsf{A} & \mathsf{C} \\ \mathsf{D} & \mathsf{E} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u} \\ \mathbf{0} \end{bmatrix} &= \begin{bmatrix} \mathbf{v}^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \mathbf{b} \\ \mathbf{e} \end{bmatrix} \\ \begin{bmatrix} \mathbf{v}^T & \mathbf{0}^T \end{bmatrix} \cdot \begin{bmatrix} \mathsf{A} \cdot \mathbf{u} + \mathsf{C} \cdot \mathbf{0} \\ \mathsf{D} \cdot \mathbf{u} + \mathsf{E} \cdot \mathbf{0} \end{bmatrix} &= \begin{bmatrix} \mathbf{v}^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \mathbf{b} \\ \mathbf{e} \end{bmatrix} \\ \mathbf{v}^T \cdot \mathsf{A} \cdot \mathbf{u} + \mathbf{0}^T \cdot \mathsf{D} \cdot \mathbf{u} &= \mathbf{v}^T \cdot \mathbf{b} + \mathbf{0}^T \cdot \mathbf{e} \\ \mathbf{v}^T \cdot \mathsf{A} \cdot \mathbf{u} &= \mathbf{v}^T \cdot \mathbf{b} \end{aligned}$$

Como esta igualdad debe valer $\forall \mathbf{v}$, entonces $\mathsf{A} \cdot \mathbf{u} - \mathbf{b} = 0$.

□

Corolario 3.9. *Si* $\tilde{\mathbf{v}} = \begin{bmatrix} \mathbf{v} & \mathbf{0} \end{bmatrix}^T$ *y* $\tilde{\mathbf{u}} = \begin{bmatrix} \mathbf{u} & \mathbf{0} \end{bmatrix}^T$ *entonces el contenido de las matrices* C, D *y* E *y del vector* $\mathbf{e}$ *es irrelevante.*

Teorema 3.13. *La matriz* $\tilde{\mathsf{A}}$ *es singular. Más aún,* $\ker(\tilde{\mathsf{A}}) = 1$.

Corolario 3.10. *Sean los objetos*

$$\mathsf{K} = \begin{bmatrix} \mathsf{A} & \mathsf{C} \\ 0 & \mathsf{I} \end{bmatrix} \quad \mathbf{f} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix} \tag{3.38}$$

tales que $\mathsf{A} \cdot \mathbf{u} = \mathbf{b}$, *donde* I *es la matriz identidad de tamaño* $J_D \times J_D$. *Entonces el vector* $\boldsymbol{\varphi}$ *tal que* $\mathsf{K} \cdot \boldsymbol{\varphi} = \mathbf{f}$ *es igual a*

$$\boldsymbol{\varphi} = \begin{bmatrix} \mathbf{u} \\ \mathbf{0} \end{bmatrix}$$

Demostración. Sea $\boldsymbol{\varphi} = \begin{bmatrix} \boldsymbol{\varphi}_1 & \boldsymbol{\varphi}_2 \end{bmatrix}^T$. Entonces $\mathsf{K} \cdot \boldsymbol{\varphi}$ es

$$\begin{bmatrix} \mathsf{A} & \mathsf{C} \\ 0 & \mathsf{I} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{\varphi}_1 \\ \boldsymbol{\varphi}_2 \end{bmatrix} = \begin{bmatrix} \mathsf{A} \cdot \boldsymbol{\varphi}_1 + \mathsf{C} \cdot \boldsymbol{\varphi}_2 \\ 0 \cdot \boldsymbol{\varphi}_1 + \mathsf{I} \cdot \boldsymbol{\varphi}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}$$

De la segunda fila se tiene $\boldsymbol{\varphi}_2 = \mathbf{0}$. Reemplazando este resultado en la primera fila, $\mathsf{A} \cdot \boldsymbol{\varphi}_1 = \mathbf{b}$. Luego $\boldsymbol{\varphi}_1 = \mathsf{A}^{-1} \cdot \mathbf{b} = \mathbf{u}$.

□

La importancia de este resultado radica en que si pudiésemos construir la matriz extendida $\tilde{\mathsf{A}} \in \mathbb{R}^{(J+J_D)\times(J+J_D)}$ donde el elemento de la fila i y la columna j es

$$\tilde{a}_{ij} = a\Big(h_i(\mathbf{x}), h_j(\mathbf{x})\Big) \quad \text{para } i = 1, \dots, J + J_D \text{ y } j = 1, \dots, J + J_D$$

sin distinguir entre nodos en U, en Γ_N o en Γ_D, entonces podríamos obtener la matriz K reemplazando las filas correspondientes a $i = J + 1, \dots, J + J_D$ por todos ceros, excepto un uno (o cualquier valor $\alpha \neq 0$) en la diagonal. Al mismo tiempo, tendríamos que reemplazar los elementos del vector $\mathbf{f}$

$$f_j = \mathcal{B}\Big(h_j(\mathbf{x})\Big) \quad \text{para } J = 1, \dots, J + J_D$$

para $j > J$ por $f_j = 0$.

Definición 3.19 (matriz de rigidez). La matriz cuadrada K de tamaño igual a la cantidad total $J + J_D$ de nodos tal que $\mathsf{K} \cdot \boldsymbol{\varphi} = \mathbf{f}$ se llama (usualmente) *matriz de rigidez*.

Teorema 3.14. *La matriz de rigidez tiene inversa.*

Demostración. La matriz K tiene una estructura de bloque triangular superior

$$\mathsf{K} = \begin{bmatrix} \mathsf{A} & \mathsf{C} \\ 0 & \mathsf{I} \end{bmatrix}$$

Luego su determinante es

$$\det(\mathsf{K}) = \det(\mathsf{A}) \cdot \det(\mathsf{I}) = \det(\mathsf{A}) \neq 0$$

ya que A es definida positiva por el teorema 3.8.

□

Observación. Aún cuando la matriz A sea simétrica, la matriz de rigidez K (ecuación 3.38) no lo es. Sin embargo, es posible realizar el procedimiento de reemplazar filas por ceros excepto en la diagonal agregando operaciones extra de reemplazo de columnas por ceros excepto en la diagonal mientras al mismo tiempo se realizan operaciones equivalentes sobre el vector $\mathbf{f}$ del miembro derecho de forma tal de obtener un sistema de ecuaciones equivalente donde la matriz sea simétrica. Estos detalles forman parte de la implementación computacional y no de la teoría detrás del método numérico.

Observación. El procedimiento propuesto para obtener la matriz de rigidez no es el único. Otra formas de incorporar las condiciones de Dirichlet a la matriz de rigidez incluyen [18]

1. Eliminación directa
2. Método de penalidad
3. Multiplicadores de Lagrange

De todas maneras, este procedimiento…

a. es computacionalmente eficiente (especialmente si se elige la constante $\alpha \neq 0$ que se pone en la diagonal de las filas de Dirichlet en forma apropiada y se mantiene la simetría de la matriz del sistema de ecuaciones), y
b. permite incorporar condiciones de contorno de Dirichlet no homogéneas de una forma muy natural como mostramos a continuación.

En efecto, supongamos ahora que tengamos que resolver el problema de la ecuación 3.26 con una condición de contorno de Dirichlet no homogénea

$$u(\mathbf{x}) = g(\mathbf{x}) \quad \forall \mathbf{x} \in \Gamma_D$$

La forma de resolver el problema continuo que introdujimos en la sección 3.4.1.2 fue considerar $u_g \in H^1_g$, escribir $u_h = u - u_g$ y encontrar $u_h \in V$ tal que

$$a(u_h, v) = \mathcal{B}(v) - a(u_g, v) \quad \forall v \in V$$

Ahora,

1. volvemos a pasar $a(u_g, v)$ al miembro izquierdo aprovechando la bi-linealidad de a

$$a(u_h + u_g, v) = a(u, v) = \mathcal{B}(v)$$

2. escribimos la parte homogénea u_h como

$$u_h(\mathbf{x}) = \sum_{j=1}^{J} h_j(\mathbf{x}) \cdot u_j + \sum_{j=J+1}^{J+J_D} h_j(\mathbf{x}) \cdot 0 = \tilde{\mathsf{H}} \cdot \begin{bmatrix} \mathbf{u} \\ \mathbf{0} \end{bmatrix}$$

3. la función auxiliar u_g que satisface la condición de Dirichlet como

$$u_g(\mathbf{x}) = \sum_{j=1}^{J} h_j(\mathbf{x}) \cdot 0 + \sum_{j=J+1}^{J+J_D} h_j(\mathbf{x}) \cdot g(\mathbf{x}_j) = \tilde{\mathsf{H}} \cdot \begin{bmatrix} \mathbf{0} \\ \mathbf{g} \end{bmatrix}$$

4. y la suma $u = u_h + u_g$

$$u(\mathbf{x}) = u_h(\mathbf{x}) + u_g(\mathbf{x}) = \sum_{j=1}^{J} h_j(\mathbf{x}) \cdot u_j + \sum_{j=J+1}^{J+J_D} h_j(\mathbf{x}) \cdot g(\mathbf{x}_j) = \tilde{\mathsf{H}} \cdot \tilde{\mathbf{u}}$$

   donde ahora extendemos $\mathbf{u}$ como

$$\tilde{\mathbf{u}} = \begin{bmatrix} \mathbf{u} \\ \mathbf{g} \end{bmatrix}$$

   y

$$\mathbf{g} = \begin{bmatrix} g(\mathbf{x}_{J+1}) \\ g(\mathbf{x}_{J+2}) \\ \vdots \\ g(\mathbf{x}_{J+J_D}) \end{bmatrix}$$

Como $v(\mathbf{x}) \in V_J \subset H_0^1$, entonces $\mathbf{v}$ todavía se extiende con ceros

$$\tilde{\mathbf{v}} = \begin{bmatrix} \mathbf{v} \\ \mathbf{0} \end{bmatrix}$$

y el problema discretizado queda

$$\begin{bmatrix} \mathbf{v}^T & \mathbf{0}^T \end{bmatrix} \cdot \begin{bmatrix} \mathsf{A} & \mathsf{C} \\ \mathsf{D} & \mathsf{E} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u} \\ \mathbf{g} \end{bmatrix} = \begin{bmatrix} \mathbf{v}^T & \mathbf{0}^T \end{bmatrix} \cdot \begin{bmatrix} \mathbf{b} \\ \mathbf{e} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{v}^T & \mathbf{0}^T \end{bmatrix} \cdot \begin{bmatrix} \mathsf{A} \cdot \mathbf{u} + \mathsf{C} \cdot \mathbf{g} \\ \mathsf{D} \cdot \mathbf{u} + \mathsf{E} \cdot \mathbf{g} \end{bmatrix} = \begin{bmatrix} \mathbf{v}^T & \mathbf{0}^T \end{bmatrix} \cdot \begin{bmatrix} \mathbf{b} \\ \mathbf{e} \end{bmatrix}$$

$$\mathbf{v}^T \cdot \mathsf{A} \cdot \mathbf{u} + \mathbf{v}^T \cdot \mathsf{C} \cdot \mathbf{g} = \mathbf{v}^T \cdot \mathbf{b}$$

para todo $\mathbf{u} \in \mathbb{R}^J$. Es decir, la aproximación de Galerkin para el problema con condiciones de Dirichlet no homogéneas es

$$\mathsf{A} \cdot \mathbf{u} + \mathsf{C} \cdot \mathbf{g} = \mathbf{b} \tag{3.39}$$

Corolario 3.11. *Sean los objetos*

$$\mathsf{K} = \begin{bmatrix} \mathsf{A} & \mathsf{C} \\ 0 & \mathsf{I} \end{bmatrix} \quad \mathbf{f} = \begin{bmatrix} \mathbf{b} \\ \mathbf{g} \end{bmatrix}$$

tales que se satisface la ecuación 3.39, entonces el vector $\boldsymbol{\varphi}$ tal que $\mathsf{K} \cdot \boldsymbol{\varphi} = \mathbf{f}$ es igual a

$$\boldsymbol{\varphi} = \begin{bmatrix} \mathbf{u} \\ \mathbf{g} \end{bmatrix}$$

Demostración. Sea $\boldsymbol{\varphi} = \begin{bmatrix} \boldsymbol{\varphi}_1 & \boldsymbol{\varphi}_2 \end{bmatrix}^T$. Entonces $\mathsf{K} \cdot \boldsymbol{\varphi}$ es

$$\begin{bmatrix} \mathsf{A} & \mathsf{C} \\ 0 & \mathsf{I} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{\varphi}_1 \\ \boldsymbol{\varphi}_2 \end{bmatrix} = \begin{bmatrix} \mathsf{A} \cdot \boldsymbol{\varphi}_1 + \mathsf{C} \cdot \boldsymbol{\varphi}_2 \\ 0 \cdot \boldsymbol{\varphi}_1 + \mathsf{I} \cdot \boldsymbol{\varphi}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{g} \end{bmatrix}$$

De la segunda fila se tiene $\boldsymbol{\varphi}_2 = \mathbf{g}$. Reemplazando este resultado en la primera fila,

$$\mathsf{A} \cdot \boldsymbol{\varphi}_1 + \mathsf{C} \cdot \mathbf{g} = \mathbf{b}$$

Dado que se satisface la ecuación 3.39 entonces $\boldsymbol{\varphi}_1 = \mathbf{u}$.

□

Observación. Los corolarios 3.10 y 3.11 muestran que el procedimiento de reemplazar las filas correspondientes a los puntos de Γ_D por ceros excepto un uno (o $\alpha \neq 0$) en la diagonal y por el valor de la condición de contorno $g(\mathbf{x}_j)$ (o $\alpha \cdot g(\mathbf{x}_j)$) en dicho punto funciona tanto para condiciones homogéneas como no homogéneas.

Observación. Para el caso no homogéneo, el contenido de la matriz C que contiene el resultado de aplicar el operador a entre las funciones de forma del interior de U y de Γ_N contra las funciones de forma de Γ_D

$$c_{i,j-J} = a\left(h_i(\mathbf{x}), h_j(\mathbf{x})\right) \quad \text{para } i = 1, \ldots, J \text{ y } j = J+1, J+J_D$$

no es irrelevante como lo era para el caso $g(\mathbf{x}) = 0$.

Estamos entonces en condiciones resolver el primero de los dos puntos de la página 120: ¿cómo encontramos $J+J_D$ funciones de forma apropiadas? Para ello, consideramos la topología subyacente en los $J + J_D$ puntos. Tomemos la figura 3.9, que muestra no sólo los J_D puntos sobre Γ_D sino también los triángulos que forman los $J + J_D$ puntos.

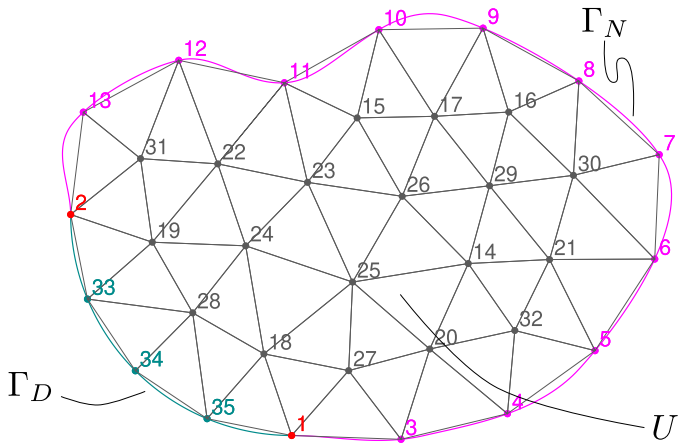


Figura 3.9.: El dominio espacial U de la figura 3.7 con $J_D = 3$ puntos extra en la frontera Γ_D para $j = 33, 34, 35$. Además, se muestran los triángulos que conectan los puntos entre sí y que cubren el dominio U.

Definición 3.20 ((laxa de) elemento). Un *elemento* es una entidad topológica de dimensión $D = 0, 1, 2$ o 3 capaz de cubrir un dominio espacial $U \in \mathbb{R}^D$.

Definición 3.21 (nodo). Llamamos a cada uno de los puntos que define un elemento, *nodo*.

Definición 3.22 (valor nodal). Dado que $h_j(\mathbf{x}_i) = \delta_{ij}$ entonces los coeficientes $v^{(j)}$ son iguales a la función v evaluada en $\mathbf{x}_j$, es decir el valor que toma la función en el nodo j

$$v^{(j)} = v(\mathbf{x}_j)$$

Llamamos a $v^{(j)}$, el *valor nodal* de la solución aproximada.

Observación. Los J valores nodales u_j son las incógnitas que se obtienen al resolver el problema numéricamente. Pero la solución al problema de Galerkin no es simplemente un conjunto de coeficientes sino una función continua $u_N(\mathbf{x})$ que puede ser evaluada en cualquier punto del espacio $\mathbf{x} \in U$.

Corolario 3.12. *Para que sea posible recuperar exactamente una función constante $u(\mathbf{x}) = cte \in U$ a partir de valores nodales constantes $u_j = cte$ las funciones de forma deben sumar uno $\forall \mathbf{x} \in U$. En resumen, las funciones de forma deben cumplir*

$$\begin{cases} h_j(\mathbf{x}_i) = \delta_{ij} & \textit{para} \quad j = 1, \dots, J + J_D \quad \textit{e} \quad i = 1, \dots, J + J_D \\ \sum_{j=1}^{J+J_D} h_j(\mathbf{x}) = 1 & \forall \mathbf{x} \in U \textit{ (incluyendo la frontera } \partial U) \end{cases} \tag{3.40}$$

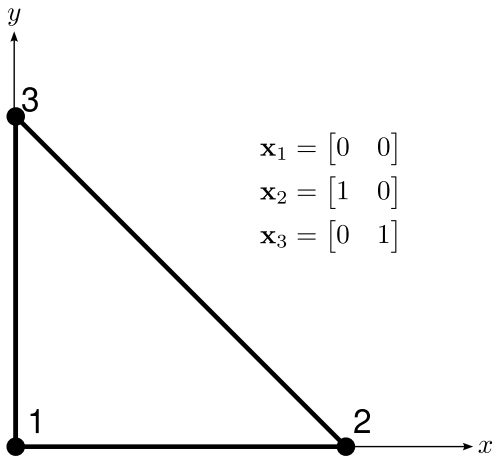
Si los elementos son apropiados, la integral sobre el dominio U es aproximadamente igual a la suma de las integrales sobre cada uno de los I elementos e_1, e_2, ..., e_I en los que lo dividimos. De hecho, los elementos son "apropiados" justamente si a medida que dividimos el dominio en más y más elementos cada vez de menor tamaño (lo que implica que $J \to \infty$), entonces

$$\lim_{I\to\infty} \sum_{i=1}^{I} \int_{e_i} f(\mathbf{x})\, d^D\mathbf{x} = \int_U f(\mathbf{x})\, d^D\mathbf{x}$$

para cualquier función $f(\mathbf{x}) : U \mapsto \mathbb{R}$ integrable.

La idea básica del método de elementos finitos (al menos para problemas lineales) es justamente concentrarse en escribir las integrales que definen la matriz de rigidez y el vector del miembro derecho en cada uno de los elementos e_i para luego "ensamblar" estos objetos globales a partir de las contribuciones elementales. Justamente, este proceso de enfocarse en los elementos es muy eficiente desde del punto de vista computacional ya que se presta perfectamente para ser realizado en forma paralela como mostramos en el capítulo 4.

Para fijar ideas, supongamos por un momento que tenemos el siguiente elemento triangular en el plano x-y:



Consideremos las funciones

$$\begin{aligned} h_1(\mathbf{x}) &= 1 - x - y \\ h_2(\mathbf{x}) &= x \\ h_3(\mathbf{x}) &= y \end{aligned}$$

Si el triángulo fuese el dominio $U \in \mathbb{R}^2$ y quisiéramos resolver una ecuación diferencial en derivadas parciales discretizándolo con los tres nodos $\mathbf{x}_1$, $\mathbf{x}_2$ y $\mathbf{x}_3$ entonces estas funciones de forma cumplirían los requerimientos de la ecuación 3.40. Recordando las ecuaciones 3.35 y 3.36,

$$\begin{aligned} \tilde{\mathsf{H}}(\mathbf{x}) &= \begin{bmatrix}1-x-y & x & y\end{bmatrix} \quad \mathbb{R}^{1\times J} \\ \tilde{\mathsf{B}}(\mathbf{x}) &= \begin{bmatrix}-1 & +1 & 0 \\ -1 & 0 & +1\end{bmatrix} \quad \mathbb{R}^{D\times J} \end{aligned}$$

Usando las expresiones de la ecuación 3.34, podemos calcular explícitamente la matriz aumentada $\tilde{\mathsf{A}}$ como

$$\tilde{\mathsf{A}} = \int_e \begin{bmatrix} -1 & -1 \\ +1 & 0 \\ 0 & +1 \end{bmatrix} \cdot k(\mathbf{x}) \cdot \begin{bmatrix} -1 & +1 & 0 \\ -1 & 0 & +1 \end{bmatrix} d^D\mathbf{x} = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \cdot \int_e f(\mathbf{x})\, d^D\mathbf{x}$$

donde e se refiere al elemento triangular. Si $f(\mathbf{x}) = 1$, entonces la integral es el área del triángulo, que es $1/2$. Lo importante del ejemplo es que la matriz de rigidez elemental

1. es cuadrada de tamaño $J \times J$ siendo J el número de nodos del elemento,[18] y
2. conocidas las funciones de forma del elemento, podemos calcular fácilmente primero derivando las h_j con respecto a las variables espaciales y luego integrando la misma expresión de la matriz extendida global sobre el elemento.

En este caso en particular, dado que las funciones de forma son lineales con respecto a las variables espaciales entonces la matriz $\mathsf{B}(\mathbf{x})$ es uniforme y puede salir fuera de la integral. Para otras topologías de elementos (por ejemplo cuadrángulos) o para elementos de órdenes superiores (en los que se agregan nodos sobre los lados o sobre el seno del elemento), las funciones de forma tendrán una dependencia más compleja y sus derivadas dependerán de $\mathbf{x}$ por lo que efectivamente habrá que integrar el producto $\mathsf{B}^T(\mathbf{x})k(\mathbf{x})\mathsf{B}(\mathbf{x})$ sobre el triángulo. Si bien en general es posible utilizar cualquier método de cuadratura numérica (incluyendo métodos adaptativos), la forma usual de calcular estas integrales es utilizando el método de integración de Gauss que consiste en disponer de una cantidad pre-fijada Q de pares de pesos ω_q y puntos espaciales $\mathbf{x}_q$ tales que

$$\int_e \mathbf{F}(\mathbf{x})\, d^D\mathbf{x} \approx \sum_{q=1}^{Q} \omega_q \cdot \mathbf{F}(\mathbf{x}_q)$$

donde el número Q depende de la precisión de la aproximación: mientras mayor sea Q, mayor será la precisión de la integral (y mayor será el costo computacional para calcularla).

Está claro que los elementos triangulares de la figura 3.7 no coinciden con el triángulo canónico de vértices $[0, 0]$, $[1, 0]$ y $[0, 1]$. Pero podemos suponer que este elemento canónico e_c, cuya matriz elemental ya sabemos calcular, vive en un plano bidimensional ξ-η. Si pudiésemos encontrar, para cada elemento real e_i del dominio U, una transformación biyectiva entre las coordenadas reales x-y y las coordenadas canónicas ξ-η entonces podríamos calcular por un lado las integrales utilizando el jacobiano J de la transformación $\mathbf{x} \mapsto \boldsymbol{\xi}$

$$\int_{e_i} f(\mathbf{x})\, d^2\mathbf{x} = \int_{e_c} f(\boldsymbol{\xi}) \cdot \left|\det(\mathsf{J})\right| d^2\boldsymbol{\xi}$$

y por otro las derivadas con respecto a las coordenadas originales x-y que aparezcan en los integrandos utilizando la regla de la cadena

[18] Para un problema con $G > 1$ grados de libertad por nodo el tamaño es $GJ \times GJ$.

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial \xi} \cdot \frac{\partial \xi}{\partial x} + \frac{\partial f}{\partial \eta} \cdot \frac{\partial \eta}{\partial x}$$
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial \xi} \cdot \frac{\partial \xi}{\partial y} + \frac{\partial f}{\partial \eta} \cdot \frac{\partial \eta}{\partial y}$$

Para ello consideremos la siguiente transformación inversa de $\boldsymbol{\xi}$ a $\mathbf{x}$

$$x = h_1(\boldsymbol{\xi}) \cdot x_1 + h_2(\boldsymbol{\xi}) \cdot x_2 + h_3(\boldsymbol{\xi}) \cdot x_3 = \sum_{j=1}^{3} h_j(\boldsymbol{\xi}) \cdot x_j$$
$$y = h_1(\boldsymbol{\xi}) \cdot y_1 + h_2(\boldsymbol{\xi}) \cdot y_2 + h_3(\boldsymbol{\xi}) \cdot y_3 = \sum_{j=1}^{3} h_j(\boldsymbol{\xi}) \cdot y_j$$

donde x_j e y_j son las coordenadas del nodo j-ésimo del elemento triangular e_i. Es decir, la transformación (inversa) propuesta consiste en interpolar las coordenadas reales continuas $\mathbf{x}$ a partir de las coordenadas reales $\mathbf{x}_j$ de los nodos del elemento real e_i usando las funciones de forma del elemento canónico e_c.[19] Las derivadas parciales de x e y con respecto a ξ y η son

$$\begin{aligned}
\frac{\partial x}{\partial \xi} &= \sum_{j=1}^{3} \frac{\partial h_j}{\partial \xi} \cdot x_j \\
\frac{\partial x}{\partial \eta} &= \sum_{j=1}^{3} \frac{\partial h_j}{\partial \eta} \cdot x_j \\
\frac{\partial y}{\partial \xi} &= \sum_{j=1}^{3} \frac{\partial h_j}{\partial \xi} \cdot y_j \\
\frac{\partial y}{\partial \eta} &= \sum_{j=1}^{3} \frac{\partial h_j}{\partial \eta} \cdot y_j
\end{aligned} \tag{3.41}$$

Los diferenciales dx y dy se relacionan con los diferenciales $d\xi$ y $d\eta$ como

$$dx = \frac{\partial x}{\partial \xi} \cdot d\xi + \frac{\partial x}{\partial \eta} \cdot d\eta$$
$$dy = \frac{\partial y}{\partial \xi} \cdot d\xi + \frac{\partial y}{\partial \eta} \cdot d\eta$$

que en forma matricial podemos escribir como

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} \cdot \begin{bmatrix} d\xi \\ d\eta \end{bmatrix} = \mathsf{J} \cdot \begin{bmatrix} d\xi \\ d\eta \end{bmatrix} \tag{3.42}$$

[19] Cuando las coordenadas se interpolan con las funciones de forma del elemento estamos ante elementos *isoparamétricos*. Existen también elementos sub-paramétricos y super-paramétricos. En este trabajo, siempre utilizamos elementos isoparamétricos.

De la misma manera,

$$\begin{bmatrix} d\xi \\ d\eta \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} dx \\ dy \end{bmatrix}$$

Reemplazando en la ecuación 3.42

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} dx \\ dy \end{bmatrix}$$

Por lo tanto, debe ser

$$\begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

y entonces

$$\mathsf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} \quad \mathsf{J}^{-1} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} \end{bmatrix}$$

Teorema 3.15. *Para que una transformación* $\mathbf{x} \mapsto \boldsymbol{\xi}$ *sea biyectiva, es decir haya una correspondencia uno a uno entre* $\mathbf{x}$ *y* $\boldsymbol{\xi}$*, el determinante del jacobiano* J *debe mantener su signo en el dominio de definición de la transformación.*

Corolario 3.13. *Si la transformación* $\mathbf{x} \mapsto \boldsymbol{\xi}$ *es biyectiva,* $\det(\mathsf{J}) \neq 0$ *y* J *tiene inversa.*

En forma análoga a la ecuación 3.41, en un elemento e_i el gradiente de una función $v \in V$ es

$$\begin{bmatrix} \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial h_1}{\partial x} & \frac{\partial h_2}{\partial x} & \frac{\partial h_3}{\partial x} \\ \frac{\partial h_1}{\partial y} & \frac{\partial h_2}{\partial y} & \frac{\partial h_3}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} v(\mathbf{x_1}) \\ v(\mathbf{x_2}) \\ v(\mathbf{x_3}) \end{bmatrix}$$

Según la regla de la cadena,

$$\frac{\partial h_1}{\partial x} = \frac{\partial h_1}{\partial \xi} \cdot \frac{\partial \xi}{\partial x} + \frac{\partial h_1}{\partial \eta} \cdot \frac{\partial \eta}{\partial x} = \begin{bmatrix} \frac{\partial h_1}{\partial \xi} & \frac{\partial h_1}{\partial \eta} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial \xi}{\partial x} \\ \frac{\partial \eta}{\partial x} \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial h_1}{\partial \xi} \\ \frac{\partial h_1}{\partial \eta} \end{bmatrix}$$

Entonces podemos escribir

$$\begin{bmatrix} \dfrac{\partial v}{\partial x} \\ \dfrac{\partial v}{\partial y} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \xi}{\partial x} & \dfrac{\partial \eta}{\partial x} \\ \dfrac{\partial \xi}{\partial y} & \dfrac{\partial \eta}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \dfrac{\partial h_1}{\partial \xi} & \dfrac{\partial h_2}{\partial \xi} & \dfrac{\partial h_3}{\partial \xi} \\ \dfrac{\partial h_1}{\partial \eta} & \dfrac{\partial h_2}{\partial \eta} & \dfrac{\partial h_3}{\partial \eta} \end{bmatrix} \cdot \begin{bmatrix} v(\mathbf{x_1}) \\ v(\mathbf{x_2}) \\ v(\mathbf{x_3}) \end{bmatrix} = \mathsf{J}^{-T}(\boldsymbol{\xi}) \cdot \mathsf{B}_c(\boldsymbol{\xi}) \cdot \mathbf{v} \tag{3.43}$$

donde hemos introducido la matriz B_c del elemento canónico

$$\mathsf{B}_c(\boldsymbol{\xi}) = \begin{bmatrix} \dfrac{\partial h_1}{\partial \xi} & \dfrac{\partial h_2}{\partial \xi} & \dfrac{\partial h_3}{\partial \xi} \\ \dfrac{\partial h_1}{\partial \eta} & \dfrac{\partial h_2}{\partial \eta} & \dfrac{\partial h_3}{\partial \eta} \end{bmatrix}$$

tal que

$$\mathsf{B}(\boldsymbol{\xi}) = \mathsf{J}^{-T}(\boldsymbol{\xi}) \cdot \mathsf{B}_c(\boldsymbol{\xi}) \tag{3.44}$$

Más aún, consideremos un triángulo en el plano x-y de coordenadas

$$\mathbf{x}_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix}$$

tal que podamos construir una matriz de coordenadas C

$$\mathsf{C} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}$$

Entonces las ecuaciones 3.41 escritas en forma matricial quedan

$$\begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix} \cdot \begin{bmatrix} \dfrac{\partial h_1}{\partial \xi} & \dfrac{\partial h_1}{\partial \eta} \\ \dfrac{\partial h_2}{\partial \xi} & \dfrac{\partial h_2}{\partial \eta} \\ \dfrac{\partial h_3}{\partial \xi} & \dfrac{\partial h_3}{\partial \eta} \end{bmatrix}$$

Es decir, podemos calcular el jacobiano J de la transformación $\boldsymbol{\xi} \mapsto \mathbf{x}$ como

$$\mathsf{J}(\boldsymbol{\xi}) = \mathsf{C} \cdot \mathsf{B}_c^T(\boldsymbol{\xi}) \tag{3.45}$$

y

$$\mathsf{J}^{-T}(\boldsymbol{\xi}) = \left[\mathsf{B}_c^T(\boldsymbol{\xi}) \cdot \mathsf{C}^T\right]^{-1} \tag{3.46}$$

por lo que

$$\mathsf{B}(\boldsymbol{\xi}) = \left[\mathsf{B}_c^T(\boldsymbol{\xi}) \cdot \mathsf{C}^T\right]^{-1} \cdot \mathsf{B}_c(\boldsymbol{\xi})$$

Observación. Si el problema es tri-dimensional, el elemento canónico e_{c_i} es el tetraedro de dimensión $D = 3$ cuyos $J_i = 4$ vértices tienen coordenadas $\boldsymbol{\xi}_1 = [0, 0, 0]$, $\boldsymbol{\xi}_2 = [1, 0, 0]$, $\boldsymbol{\xi}_3 = [0, 1, 0]$ y $\boldsymbol{\xi}_4 = [0, 0, 1]$. Entonces

a. las $J_i = 4$ funciones de forma son

$$\begin{aligned} h_1(\xi, \eta, \zeta) &= 1 - \xi - \eta - \zeta \\ h_2(\xi, \eta, \zeta) &= \xi \\ h_3(\xi, \eta, \zeta) &= \eta \\ h_4(\xi, \eta, \zeta) &= \zeta \end{aligned}$$

b. la transformación $\boldsymbol{\xi} \in \mathbb{R}^3 \mapsto \mathbf{x} \in \mathbb{R}^3$ es

$$\begin{aligned} x &= \sum_{j=1}^{4} h_j(\boldsymbol{\xi}) \cdot x_j \\ y &= \sum_{j=1}^{4} h_j(\boldsymbol{\xi}) \cdot y_j \\ z &= \sum_{j=1}^{4} h_j(\boldsymbol{\xi}) \cdot z_j \end{aligned}$$

c. el jacobiano $\mathsf{J} \in \mathbb{R}^{3\times3}$ y su inversa $\mathsf{J}^{-1} \in \mathbb{R}^{3\times3}$ son

$$\mathsf{J} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} & \dfrac{\partial x}{\partial \zeta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} & \dfrac{\partial y}{\partial \zeta} \\ \dfrac{\partial z}{\partial \xi} & \dfrac{\partial z}{\partial \eta} & \dfrac{\partial z}{\partial \zeta} \end{bmatrix} \quad \mathsf{J}^{-1} = \begin{bmatrix} \dfrac{\partial \xi}{\partial x} & \dfrac{\partial \xi}{\partial y} & \dfrac{\partial \xi}{\partial z} \\ \dfrac{\partial \eta}{\partial x} & \dfrac{\partial \eta}{\partial y} & \dfrac{\partial \eta}{\partial z} \\ \dfrac{\partial \zeta}{\partial x} & \dfrac{\partial \zeta}{\partial y} & \dfrac{\partial \zeta}{\partial z} \end{bmatrix}$$

d. la matriz de derivadas canónicas $\mathsf{B}_c \in \mathbb{R}^{3\times4}$ es

$$\mathsf{B}_c(\boldsymbol{\xi}) = \begin{bmatrix} \dfrac{\partial h_1}{\partial \xi} & \dfrac{\partial h_2}{\partial \xi} & \dfrac{\partial h_3}{\partial \xi} & \dfrac{\partial h_4}{\partial \xi} \\ \dfrac{\partial h_1}{\partial \eta} & \dfrac{\partial h_2}{\partial \eta} & \dfrac{\partial h_3}{\partial \eta} & \dfrac{\partial h_4}{\partial \eta} \\ \dfrac{\partial h_1}{\partial \zeta} & \dfrac{\partial h_2}{\partial \zeta} & \dfrac{\partial h_3}{\partial \zeta} & \dfrac{\partial h_4}{\partial \zeta} \end{bmatrix}$$

e. la matriz de coordenadas $\mathsf{C} \in \mathbb{R}^{3\times4}$ es

$$\mathsf{C} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix}$$

pero las relaciones

$$\begin{aligned} \nabla \mathbf{v} &= \mathsf{J}^{-T} \cdot \mathsf{B}_c \cdot \mathbf{v} && (3.43) \\ \mathsf{B} &= \mathsf{J}^{-T} \cdot \mathsf{B}_c && (3.44) \\ \mathsf{J} &= \mathsf{C} \cdot \mathsf{B}_c^T && (3.45) \\ \mathsf{J}^{-T} &= \left[\mathsf{B}_c^T \cdot \mathsf{C}\right]^{-1} && (3.46) \\ \mathsf{B} &= \left[\mathsf{B}_c^T \cdot \mathsf{C}^T\right]^{-1} \cdot \mathsf{B}_c && (3.33) \end{aligned} \tag{3.47}$$

siguen siendo válidas.

Observación. Además de triángulos (tetraedros) se podrían haber utilizado elementos cuadrangulares (hexaédricos, prismáticos o piramidales), cada uno con su correspondiente elemento canónico en el plano ξ-η (espacio ξ-η-ζ) y funciones de forma $h_j(\boldsymbol{\xi})$ para $j = 1, \dots, J$. Las relaciones matriciales 3.47 seguirían siendo válidas.

Observación. Además de elementos lineales en los que hay un nodo en cada vértice, también existen elementos de órdenes superiores con nodos en los lados y eventualmente en el seno del elemento (figura 3.10).

(a) Nodo sobre un vértice de un triángulo

(b) Nodo sobre un lado de un triángulo

Figura 3.10.: Funciones de forma de segundo orden tras agregar puntos en los bordes de los triángulos de la figura 3.9.

Observación. Los elementos del jacobiano J están dados explícitamente por las ecuaciones 3.41. En algún sentido, J es "fácil" ya que las funciones de forma $h_j(\boldsymbol{\xi})$ tienen una dependencia sencilla con $\boldsymbol{\xi}$. Por otro lado, los elementos de J^{-1} no están disponibles directamente ya que, en general, no tenemos una expresión explícita de $\boldsymbol{\xi}(\mathbf{x})$ a partir de la cual calcular las derivadas parciales. Para poder evaluar las derivadas parciales de ξ y η (y eventualmente ζ) con respecto a x e y (y eventualmente z) se debe primero calcular el jacobiano "fácil" $\mathsf{J} \in \mathbb{R}^{2\times 2}$ (eventualmente $\mathbb{R}^{3\times 3}$) con la ecuación 3.45, calcular su inversa J^{-1} explícitamente y luego, de ser necesario, extraer sus elementos uno a uno.

Para un problema de dimensión D, para cada elemento e_i del dominio discretizado $U \in \mathbb{R}^D$, una vez que conocemos

1. la topología del elemento e_i
   - segmento para $D = 1$
   - triángulo o cuadrángulo para $D = 2$

- tetraedro, hexaedro, prisma o pirámide para $D = 3$

2. las J_i funciones de forma $h_j(\boldsymbol{\xi})$ del elemento canónico e_c en el espacio $\boldsymbol{\xi} \in \mathbb{R}^D$ con las cuales construimos la matriz canónica H_c

$$\mathsf{H}_c(\boldsymbol{\xi}) = \begin{bmatrix} h_1(\boldsymbol{\xi}) & h_2(\boldsymbol{\xi}) & \cdots & h_{J_i}(\boldsymbol{\xi}) \end{bmatrix} \quad \in \mathbb{R}^{1 \times J_i}$$

3. las $J_i \cdot D$ derivadas parciales $\partial h_j / \partial \xi_d$ con respecto a las coordenadas $\boldsymbol{\xi} \in \mathbb{R}^D$, con las cuales construimos la matriz canónica B_c

$$\mathsf{B}_c(\boldsymbol{\xi}) = \begin{bmatrix} \dfrac{\partial h_1}{\partial \xi} & \dfrac{\partial h_2}{\partial \xi} & \cdots & \dfrac{\partial h_{J_i}}{\partial \xi} \\ \dfrac{\partial h_1}{\partial \eta} & \dfrac{\partial h_2}{\partial \eta} & \cdots & \dfrac{\partial h_{J_i}}{\partial \eta} \\ \dfrac{\partial h_1}{\partial \zeta} & \dfrac{\partial h_2}{\partial \zeta} & \cdots & \dfrac{\partial h_{J_i}}{\partial \zeta} \end{bmatrix} \quad \in \mathbb{R}^{D \times J_i}$$

4. el conjunto de Q pares de pesos y ubicaciones de puntos de Gauss $(\omega_q, \boldsymbol{\xi}_q)$ del elemento canónico e_c

5. las coordenadas reales $\mathbf{x}_j \in \mathbb{R}^D$ de los J_i nodos que definen el elemento real e_i con los que construimos la matriz de coordenadas C_i del elemento e_i

$$\mathsf{C}_i = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{J_i} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \cdots & x_{J_i} \\ y_1 & y_2 & \cdots & y_{J_i} \\ z_1 & z_2 & \cdots & z_{J_i} \end{bmatrix} \quad \in \mathbb{R}^{D \times J_i}$$

que permite evaluar el i-ésimo jacobiano $\mathsf{J}_i(\boldsymbol{\xi})$ como

$$\mathsf{J}_i(\boldsymbol{\xi}) = \mathsf{C}_i \cdot \mathsf{B}_c^T(\boldsymbol{\xi}) \tag{3.48}$$

y las coordenadas reales $\mathbf{x}_q$ de los Q puntos de Gauss

$$\begin{aligned} x_q &= \sum_{j=1}^{J_i} h_j(\boldsymbol{\xi}_q) \cdot x_j \\ y_q &= \sum_{j=1}^{J_i} h_j(\boldsymbol{\xi}_q) \cdot y_j \\ z_q &= \sum_{j=1}^{J_i} h_j(\boldsymbol{\xi}_q) \cdot z_j \end{aligned}$$

necesarias para evaluar $k(\mathbf{x}_q)$ y $f(\mathbf{x}_q)$ dentro de cada término de la suma de la cuadratura numérica,

entonces estamos en condiciones de evaluar la matriz $K_i \in \mathbb{R}^{J_i \times J_i}$ de rigidez elemental correspondiente al elemento e_i para la formulación en elementos finitos[20] de la ecuación generalizada de Poisson como

$$\begin{aligned}
\mathsf{K}_i &= \int_{e_i} \mathsf{B}_i^T(\mathbf{x}) \cdot k(\mathbf{x}) \cdot \mathsf{B}_i(\mathbf{x})\, d^D\mathbf{x} \\
&= \int_{e_c} \mathsf{B}_i^T(\boldsymbol{\xi}) \cdot k(\mathbf{x}_q) \cdot \mathsf{B}_i(\boldsymbol{\xi}) \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}\right)\right]\right| d^D\boldsymbol{\xi} \\
&\approx \sum_{q=1}^{Q} \omega_q \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right| \cdot \left\{\mathsf{B}_i^T(\boldsymbol{\xi}_q) \cdot k(\mathbf{x}_q) \cdot \mathsf{B}_i(\boldsymbol{\xi}_q)\right\} \\
&\approx \sum_{q=1}^{Q} \omega_q \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right| \cdot \left\{\left[\mathsf{J}_i^{-T}\left(\boldsymbol{\xi}_q\right) \cdot \mathsf{B}_c(\boldsymbol{\xi}_q)\right]^T k(\mathbf{x}_q) \left[\mathsf{J}^{-T}\left(\boldsymbol{\xi}_q\right) \cdot \mathsf{B}_c(\boldsymbol{\xi}_q)\right]\right\} \\
&\approx \sum_{q=1}^{Q} \omega_q \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right| \cdot \left\{\left[\left(\mathsf{B}^T(\boldsymbol{\xi}_q) \cdot \mathsf{C}_i^T\right)^{-1} \cdot \mathsf{B}_c(\boldsymbol{\xi}_q)\right]^T k(\mathbf{x}_q) \left[\left(\mathsf{B}_c^T(\boldsymbol{\xi}_q) \cdot \mathsf{C}_i^T\right)^{-1} \cdot \mathsf{B}_c(\boldsymbol{\xi}_q)\right]\right\}
\end{aligned}$$

y la componente volumétrica del vector elemental $\mathbf{b}_i$ como

$$\begin{aligned}
\mathbf{b}_i^{(U)} &= \int_{e_i} \mathsf{H}_c^T(\mathbf{x}) \cdot f(\mathbf{x})\, d^D\mathbf{x} \\
&= \int_{e_c} \mathsf{H}_c^T(\boldsymbol{\xi}) \cdot f(\mathbf{x}) \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}\right)\right]\right| d^D\boldsymbol{\xi} \\
&\approx \sum_{q=1}^{Q} \omega_q \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right| \cdot \left\{\mathsf{H}_c^T(\boldsymbol{\xi}_q) \cdot f(\mathbf{x}_q)\right\}
\end{aligned}$$

Definición 3.23. La matriz de rigidez elemental K_i del elemento e_i según la formulación de elementos finitos desarrollada en esta sección es

$$\mathsf{K}_i \approx \sum_{q=1}^{Q} \underbrace{\omega_q \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right|}_{\text{cuadratura numérica sobre } e_c} \underbrace{\left\{\mathsf{B}_i^T(\boldsymbol{\xi}_q) \cdot k(\mathbf{x}_q) \cdot \mathsf{B}_i(\boldsymbol{\xi}_q)\right\}}_{\text{discretización del operador } -\operatorname{div}(k \cdot \nabla u)} \tag{3.49}$$

con

i. la matriz de coordenadas C_i

$$\mathsf{C}_i = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{J_i}\end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \cdots & x_{J_i} \\ y_1 & y_2 & \cdots & y_{J_i} \\ z_1 & z_2 & \cdots & z_{J_i} \end{bmatrix} \quad \in \mathbb{R}^{D \times J_i}$$

[20] Estrictamente hablando, esta no es *la* formulación sino que es *una* de las varias formulaciones posibles. De todas maneras es la más usual y eficiente.

ii. las coordenadas reales $\mathbf{x}_q$ del punto de gauss q-ésimo

$$\mathbf{x}_q(\boldsymbol{\xi}_q) = \begin{bmatrix} \sum_{j=1}^{J_i} h_j(\boldsymbol{\xi}_q) \cdot x_j \\ \sum_{j=1}^{J_i} h_j(\boldsymbol{\xi}_q) \cdot y_j \\ \sum_{j=1}^{J_i} h_j(\boldsymbol{\xi}_q) \cdot z_j \end{bmatrix} = \sum_{j=1}^{J_i} h_j(\boldsymbol{\xi}_q) \cdot \mathbf{x}_j \quad \in \mathbb{R}^D \tag{3.50}$$

iii. la matriz canónica de funciones de forma H_c

$$\mathsf{H}_c(\boldsymbol{\xi}_q) = \begin{bmatrix} h_1(\boldsymbol{\xi}_q) & h_2(\boldsymbol{\xi}_q) & \cdots & h_1(\boldsymbol{\xi}_q) \end{bmatrix} \quad \in \mathbb{R}^{1\times J_i}$$

iv. la matriz canónica de derivadas B_c

$$\mathsf{B}_c(\boldsymbol{\xi}_q) = \begin{bmatrix} \dfrac{\partial h_1}{\partial \xi} & \dfrac{\partial h_2}{\partial \xi} & \cdots & \dfrac{\partial h_{J_i}}{\partial \xi} \\ \dfrac{\partial h_1}{\partial \eta} & \dfrac{\partial h_2}{\partial \eta} & \cdots & \dfrac{\partial h_{J_i}}{\partial \eta} \\ \dfrac{\partial h_1}{\partial \zeta} & \dfrac{\partial h_2}{\partial \zeta} & \cdots & \dfrac{\partial h_{J_i}}{\partial \zeta} \end{bmatrix} \quad \in \mathbb{R}^{D\times J_i}$$

v. el jacobiano J_i del elemento real e_i

$$\mathsf{J}_i(\boldsymbol{\xi}_q) = \mathsf{B}_c(\boldsymbol{\xi}_q) \cdot \mathsf{C}_i \quad \in \mathbb{R}^{D\times D} \tag{3.51}$$

vi. la matriz de derivadas reales B_i del elemento real e_i

$$\mathsf{B}_i(\boldsymbol{\xi}_q) = \mathsf{J}_i^{-T}(\boldsymbol{\xi}_q) \cdot \ \mathsf{B}_c(\boldsymbol{\xi}_q) = \left[\mathsf{B}_c^T(\boldsymbol{\xi}_q) \cdot \mathsf{C}_i^T\right]^{-1} \cdot \mathsf{B}_c(\boldsymbol{\xi}_q) \quad \in \mathbb{R}^{D\times J_i} \tag{3.52}$$

Definición 3.24. El vector elemental de fuentes volumétricas $\mathbf{b}_i^{(U)}$ en el elemento e_i es

$$\mathbf{b}_i^{(U)} \approx \sum_{q=1}^{Q} \underbrace{\omega_q \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right|}_{\text{cuadratura numérica sobre } e_c} \underbrace{\left\{\mathsf{H}_c^T(\boldsymbol{\xi}_q) \cdot f(\mathbf{x}_q)\right\}}_{\text{discretización del miembro derecho } f} \tag{3.53}$$

Observación. Si las funciones de forma son lineales en $\boldsymbol{\xi}$ entonces B_i es uniforme y no depende de $\boldsymbol{\xi}$. Si además $k(\mathbf{x})$ es un polinomio de orden menor o igual al orden de integración del conjunto de pesos y ubicaciones de puntos de Gauss de tamaño Q entonces la integración numérica para K_i es exacta.

Para evaluar las contribuciones de las condiciones de contorno naturales, debemos integrar sobre elementos en la frontera ∂U del dominio U. Esto es, para $D = 2$ debemos integrar sobre elementos tipo segmento que están sobre el plano x-y (pero no necesariamente sobre la recta real). Para $D = 3$ debemos integrar sobre elementos triangulares o cuadrangulares que están en el espacio x-y-z (pero no necesariamente sobre el plano x-y). Hay varias formas de atacar este problema. En esta tesis proponemos introducir una transformación intermedia desde las coordenadas $\mathbf{x} \in \mathbb{R}^D$ hacia un sistema de coordenadas $\mathbf{r} \in \mathbb{R}^{D-1}$, para luego sí transformar las coordenadas $\mathbf{r}$ a $\boldsymbol{\xi}$ y realizar la integración. Por ejemplo, si la condición de contorno de Neumann implica integrar sobre un triángulo arbitrario cuyas coordenadas son $\mathbf{x}_1$, $\mathbf{x}_2$ y $\mathbf{x}_3 \in \mathbb{R}^3$ entonces primero encontramos una (de las infinitas) rotaciones $\mathbf{x} \in \mathbb{R}^3 \mapsto \mathbf{r} \in \mathbb{R}^2$ para luego transformar $\mathbf{r} \in \mathbb{R}^2 \mapsto \boldsymbol{\xi} \in \mathbb{R}^2$ y poder usar las matrices del elemento triangular canónico.

Teorema 3.16. *La matriz de rotación que lleva el vector* $\mathbf{a} \in \mathbb{R}^3$ *al vector* $\mathbf{b} \in \mathbb{R}^3$ *es*

$$\mathsf{R} = \mathsf{I} + \mathsf{T} + \frac{1}{1 - \mathbf{a} \cdot \mathbf{b}} \cdot \mathsf{T}^2$$

donde la matriz T *es*

$$\mathsf{T} = \begin{bmatrix} 0 & -t_3 & +t_2 \\ +t_3 & 0 & -t_1 \\ -t_2 & +t_1 & 0 \end{bmatrix}$$

y $\mathbf{t}$ *es producto cruz entre* $\mathbf{a}$ *y* $\mathbf{b}$

$$\mathbf{t} = \mathbf{a} \times \mathbf{b}$$

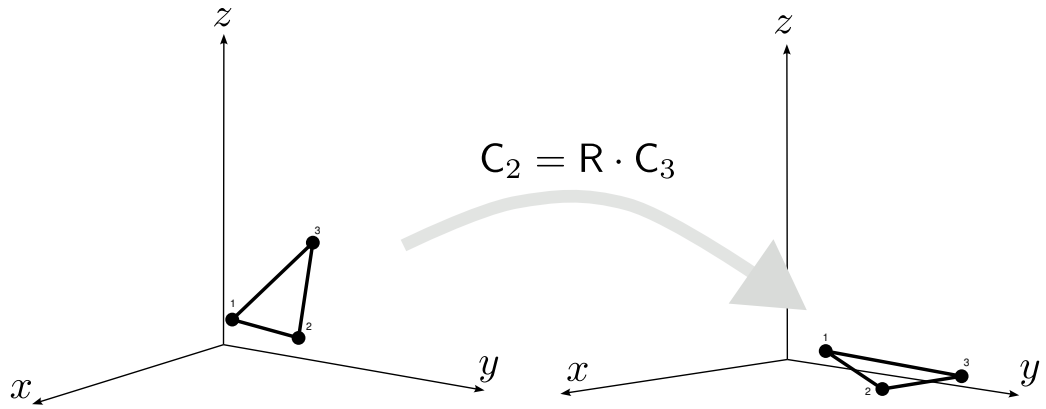


Figura 3.11.: Rotación $\mathbb{R}^3 \mapsto \mathbb{R}^3$ para que la normal a un triángulo arbitrario coincida con la dirección z.

Lo que queremos es, como ilustramos en la figura 3.11, transformar una de las dos normales $\hat{\mathbf{n}}$ del triángulo

$$\hat{\mathbf{n}} = \frac{(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)}{||(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)||}$$

para que coincida con versor normal en la dirección z, $\hat{\mathbf{e}}_z = [0, 0, 1]$. Haciendo $\mathbf{a} = \hat{\mathbf{n}}$ y $\mathbf{b} = \hat{\mathbf{e}}_z$, el vector $\mathbf{t}$ es

$$\mathbf{t} = \hat{\mathbf{n}} \times \hat{\mathbf{e}}_z = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

Con esto podemos entonces convertir la matriz con las tres coordenadas tridimensionales $\mathsf{C}_i \in \mathbb{R}^{3\times 3}$ del triángulo original a una de coordenadas bidimensionales $\mathsf{C}'_i \in \mathbb{R}^{2\times 3}$ como

$$\begin{bmatrix} \mathsf{C}'_i \\ \mathsf{0} \end{bmatrix} = \mathsf{R} \cdot \mathsf{C}_i$$

$$\begin{bmatrix} x'_1 & x'_2 & x'_3 \\ y'_1 & y'_2 & y'_3 \\ z_0 & z_0 & z_0 \end{bmatrix} = \begin{bmatrix} 1 + k \cdot (-t_3^2 - t_2^2) & -t_3 + k \cdot (t_1 \cdot t_2) & +t_2 + k \cdot (t_1 \cdot t_3) \\ +t_3 + k \cdot (t_1 \cdot t_2) & 1 + k \cdot (-t_3^2 - t_1^2) & -t_1 + k \cdot (t_2 \cdot t_3) \\ -t_2 + k \cdot (t_1 \cdot t_3) & +t_1 + k \cdot (t_2 \cdot t_3) & 1 + k \cdot (-t_2^2 - t_1^2) \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix}$$

para

$$k = \frac{1}{1 - \hat{\mathbf{n}} \cdot \hat{\mathbf{e}}_z}$$

y algún z_0 arbitrario que podemos ignorar. Entonces podemos calcular el jacobiano de un elemento de superficie $e_{i'}^{(D-1)}$ en un problema tridimensional con las funciones de forma tradicionales del triángulo canónico $e_{c'}^{(D-1)}$ cuya matriz de coordenadas C'_i es

$$\mathsf{C}'_i = \begin{bmatrix} x'_1 & x'_2 & x'_3 \\ y'_1 & y'_2 & y'_3 \end{bmatrix} \quad \in \mathbb{R}^{2\times 3}$$

Definición 3.25. La contribución de las condiciones de contorno naturales al vector $\mathbf{b}_i$ es

$$\begin{aligned} \mathbf{b}_i^{(\Gamma_N)} &= \int_{e_{i'}^{(D-1)}} \mathsf{H}_c^T(\mathbf{x}) \cdot p(\mathbf{x})\, d^{D-1}\mathbf{x} \\ &= \int_{e_{c'}^{(D-1)}} \mathsf{H}_{c'}^T(\boldsymbol{\xi}) \cdot p(\mathbf{x}) \cdot \left| \det \left[\mathsf{J}_i\left(\boldsymbol{\xi}\right) \right] \right| d^{D-1}\boldsymbol{\xi} \\ &\approx \sum_{q=1}^{Q} \omega_q^{(D-1)} \cdot \left| \det \left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right) \right] \right| \cdot \left\{ \mathsf{H}_{c'}^T(\boldsymbol{\xi}_q) \cdot p(\mathbf{x}_q) \right\} \end{aligned} \tag{3.54}$$

donde la matriz $\mathsf{H}_{c'}$ es la del elemento canónico superficial $e_{c'}^{(D-1)}$ y el jacobiano J_i es el que le corresponde al elemento superficial $e_{i'}^{(D-1)}$, ambos de dimensión $D-1$.

Observación. No tener en cuenta ningún término de superficie es equivalente a hacer que $p(\mathbf{x}) = 0$. Es decir, que las condiciones de contorno sean homogéneas en Γ_N. Para algunos problemas, como por ejemplo elasticidad, esto no presenta mayores inconvenientes ya que una condición de Neumann

homogénea sobre una superficie quiere decir que dicha superficie no tiene ninguna carga externa, que es lo que sucede en caras que están en contacto con "el vacío". Luego solamente hay que prestar atención a las superficies que tienen condiciones de contorno naturales no homogéneas. Para otros problemas, como por ejemplo conducción de calor o difusión de neutrones, una condición natural homogénea indica una superficie de simetría y no "de vacío". Hay que prestar especial atención entonces en agregar explícitamente condiciones de contorno no triviales en todas las caras expuestas al vacío, mientras que no hay que hacer nada para las caras con simetría.

3.4.2. Ecuación de difusión de neutrones

Estamos en condiciones entonces de discretizar en espacio las ecuaciones de difusión multigrupo que derivamos en la sección 3.2

$$\begin{aligned} -\text{div}\Big[D_g(\mathbf{x}) \cdot \text{grad}\left[\phi_g(\mathbf{x})\right]\Big] + \Sigma_{tg}(\mathbf{x}) \cdot \phi_g(\mathbf{x}) = \\ \sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \cdot \phi_{g'}(\mathbf{x}) + \chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \cdot \phi_{g'}(\mathbf{x}) + s_{0g}(\mathbf{x}) \end{aligned} \tag{3.8}$$

Comenzamos con el caso $G = 1$ en la sección que sigue y luego generalizamos la formulación para $G > 1$ en la sección 3.4.2.2.

3.4.2.1. Un único grupo de energía

Para $G = 1$ la ecuación se simplifica a

$$-\text{div}\Big[D(\mathbf{x}) \cdot \text{grad}\left[\phi(\mathbf{x})\right]\Big] + \Big[\Sigma_t(\mathbf{x}) - \Sigma_{s_0}(\mathbf{x}) - \nu\Sigma_f(\mathbf{x})\Big] \cdot \phi(\mathbf{x}) = s_0(\mathbf{x})$$

El término de la divergencia y el miembro derecho tienen la misma forma que la ecuación de Poisson que analizamos en la sección 3.4.1, por lo que debemos esperar contribuciones elementales $\mathtt{B}_i^T \cdot D \cdot \mathtt{B}_i$ y $\mathtt{H}_i^T \cdot s_0$ respectivamente. Para evaluar el término de fuente neta lineal con ϕ procedemos a multiplicar la formulación fuerte por una función de prueba $v(\mathbf{x}) \in V$[21] e integrar en el dominio $U \in \mathbb{R}^D$

$$\begin{aligned} \int_U v(\mathbf{x}) \cdot \Big\{-\text{div}\Big[D(\mathbf{x}) \cdot \text{grad}\left[\phi(\mathbf{x})\right]\Big]\Big\}\, d^D\mathbf{x} \\ + \int_U v(\mathbf{x}) \cdot \Big\{\Big[\Sigma_t(\mathbf{x}) - \Sigma_{s_0}(\mathbf{x}) - \nu\Sigma_f(\mathbf{x})\Big] \cdot \phi(\mathbf{x})\Big\}\, d^D\mathbf{x} = \int_U v(\mathbf{x}) \cdot s_0(\mathbf{x})\, d^D\mathbf{x} \end{aligned}$$

[21] Como para el problema de elasticidad al multiplicar la formulación fuerte por las funciones de prueba y aplicar la fórmula de Green se obtiene el principio de los trabajos virtuales, a veces estas funciones de prueba se llaman "desplazamientos virtuales". Como generalización, en el problema de conducción de calor se las llaman "temperaturas virtuales" [9]. En este caso, tal como ya propusimos en [62], podríamos llamarlas "flujos escalares virtuales".

Usando la fórmula de Green y el hecho de que $v(\mathbf{x}) = 0$ en Γ_D obtenemos la formulación débil de la ecuación de difusión para un único grupo de energía

$$\int_U \text{grad}\left[v(\mathbf{x})\right] \cdot D(\mathbf{x}) \cdot \text{grad}\left[\phi(\mathbf{x})\right]\, d^D\mathbf{x} + \int_U v(\mathbf{x}) \cdot \left[\Sigma_t(\mathbf{x}) - \Sigma_{s_0}(\mathbf{x}) - \nu\Sigma_f(\mathbf{x})\right] \cdot \phi(\mathbf{x})\, d^D\mathbf{x} = \int_U v(\mathbf{x}) \cdot s_0(\mathbf{x})\, d^D\mathbf{x} + \int_{\Gamma_N} v(\mathbf{x}) \cdot p(\mathbf{x})\, d^{D-1}\mathbf{x}$$

donde $p(\mathbf{x})$ es la condición de contorno de Neumann sobre Γ_D

$$D(\mathbf{x}) \cdot \left[\text{grad}\left[\phi(\mathbf{x})\right] \cdot \hat{\mathbf{n}}\right] = p(\mathbf{x})$$

Observación. Si la frontera Γ_D tiene una condición de simetría entonces $p(\mathbf{x}) = 0$ y las contribuciones superficiales al vector $\mathbf{b}$ son idénticamente cero.

Observación. Si la ecuación de difusión tiene una condición de vacío (definición 2.18) sobre Γ_V, entonces

$$D(\mathbf{x}) \cdot \frac{\partial\phi}{\partial n} = -\frac{1}{2} \cdot \phi(\mathbf{x})$$

Esto es equivalente a agregar el término de superficie

$$\int_{\Gamma_V} v(\mathbf{x}) \cdot \frac{1}{2} \cdot \phi(\mathbf{x})\, d^{D-1}\mathbf{x} \approx \int_{\Gamma_V} \mathbf{v}^T \cdot \mathsf{H}^T(\mathbf{x}) \cdot \frac{1}{2} \cdot \mathsf{H}^T(\mathbf{x}) \cdot \boldsymbol{\phi}\, d^{D-1}\mathbf{x}$$

al operador $a(\phi, v)$.

Observación. El operador bi-lineal $a(\phi, v) : V \times V \mapsto \mathbb{R}$ discretizado para este problema es

$$\begin{aligned}
a(\phi, v) =& \int_U \text{grad}\Big[v(\mathbf{x})\Big] D(\mathbf{x}) \text{grad}\Big[\phi(\mathbf{x})\Big]\, d^D\mathbf{x} + \int_U v(\mathbf{x}) \left[\Sigma_t(\mathbf{x}) - \Sigma_{s_0}(\mathbf{x}) - \nu\Sigma_f(\mathbf{x})\right] \phi(\mathbf{x})\, d^D\mathbf{x} \\
&+ \int_{\Gamma_V} v(\mathbf{x}) \cdot \frac{1}{2} \cdot \phi(\mathbf{x})\, d^{D-1}\mathbf{x} \\
\approx& \int_U \mathbf{v}^T \mathsf{B}^T(\mathbf{x}) D(\mathbf{x}) \mathsf{B}(\mathbf{x}) \boldsymbol{\phi}\; d^D\mathbf{x} + \int_U \mathbf{v}^T \mathsf{H}^T(\mathbf{x}) \left[\Sigma_t(\mathbf{x}) - \Sigma_{s_0}(\mathbf{x}) - \nu\Sigma_f(\mathbf{x})\right] \mathsf{H}(\mathbf{x}) \boldsymbol{\phi}\; d^D\mathbf{x} \\
&\int_{\Gamma_V} \mathbf{v}^T \cdot \mathsf{H}^T(\mathbf{x}) \cdot \frac{1}{2} \cdot \mathsf{H}^T(\mathbf{x}) \cdot \boldsymbol{\phi}\, d^{D-1}\mathbf{x} \\
=& \mathbf{v}^T \left[\int_U \mathsf{B}^T(\mathbf{x}) D(\mathbf{x}) \mathsf{B}(\mathbf{x})\, d^D\mathbf{x} + \int_U \mathsf{H}^T(\mathbf{x}) \left[\Sigma_t(\mathbf{x}) - \Sigma_{s_0}(\mathbf{x}) - \nu\Sigma_f(\mathbf{x})\right] \mathsf{H}(\mathbf{x})\, d^D\mathbf{x} \right. \\
&\left. + \int_{\Gamma_V} \mathsf{H}^T(\mathbf{x}) \cdot \frac{1}{2} \cdot \mathsf{H}^T(\mathbf{x})\, d^{D-1}\mathbf{x} \right] \boldsymbol{\phi}
\end{aligned} \tag{3.55}$$

Observación. El operador de la ecuación 3.55 es simétrico.

Observación. El operador de la ecuación 3.55 es coercitivo si $\Sigma_t(\mathbf{x}) - \Sigma_{s_0}(\mathbf{x}) > \nu\Sigma_f(\mathbf{x})$. Pero puede dejar de serlo si la desigualdad no se cumple. En efecto, la desigualdad implica $k_\infty < 1$. Siguiendo razonamientos físicos, podemos decir que el operador es coercitivo sólo si el factor de multiplicación $k_{\text{eff}} < 1$. Esto es, un medio multiplicativo crítico o super-crítico con una fuente independiente no tiene solución de estado estacionario.

Podemos escribir entonces la matriz de rigidez elemental K_i volumétrica del problema de difusión de neutrones a un grupo de energías como

$$\mathsf{K}_i^{(U)} = \sum_{q=1}^{Q} \omega_q \cdot \left| \det \left[\mathsf{J}_i \left(\boldsymbol{\xi}_q \right) \right] \right| \cdot \left[\mathsf{L}_i(\boldsymbol{\xi}_q) + \mathsf{A}_i(\boldsymbol{\xi}_q) - \mathsf{F}_i(\boldsymbol{\xi}_q) \right]$$

donde tenemos la matriz elemental de "pérdidas"[22]

$$\mathsf{L}_i = \mathsf{B}_i^T(\boldsymbol{\xi}_q) \cdot D(\boldsymbol{\xi}_q) \cdot \mathsf{B}_i^T(\boldsymbol{\xi}_q)$$

la matriz elemental de absorciones

$$\mathsf{A}_i = \mathsf{H}_c^T(\boldsymbol{\xi}_q) \cdot \left[\Sigma_t(\boldsymbol{\xi}_q) - \Sigma_{s_0}(\boldsymbol{\xi}_q) \right] \cdot \mathsf{H}_c(\boldsymbol{\xi}_q)$$

y la matriz elemental de fisiones

$$\mathsf{F}_i = \mathsf{H}_c^T(\boldsymbol{\xi}_q) \cdot \nu\Sigma_f(\boldsymbol{\xi}_q) \cdot \mathsf{H}_c(\boldsymbol{\xi}_q)$$

Observación. El funcional $\mathcal{B}(v) : V \mapsto \mathbb{R}$ es

$$\begin{aligned} \mathcal{B}(v) &= \int_U v(\mathbf{x}) \cdot s_0(\mathbf{x})\, d^D\mathbf{x} + \int_{\Gamma_N} v(\mathbf{x}) \cdot p(\mathbf{x})\, d^{D-1}\mathbf{x} \\ &= \int_U \mathbf{v}^T \cdot \mathsf{H}^T(\mathbf{x}) \cdot s_0(\mathbf{x})\, d^D\mathbf{x} + \int_{\Gamma_N} \mathbf{v}^T \cdot \mathsf{H}^T(\mathbf{x}) \cdot p(\mathbf{x})\, d^{D-1}\mathbf{x} \\ &= \mathbf{v}^T \cdot \left[\int_U \mathsf{H}^T(\mathbf{x}) \cdot s_0(\mathbf{x})\, d^D\mathbf{x} + \int_{\Gamma_N} \mathsf{H}^T(\mathbf{x}) \cdot p(\mathbf{x})\, d^{D-1}\mathbf{x} \right] \end{aligned} \tag{3.56}$$

Las contribuciones volumétricas y superficiales al vector $\mathbf{b}_i$ son similares al caso del problema de Poisson

[22] Del inglés *leakage*.

$$\begin{aligned}\mathbf{b}_i^{(U)} &= \int_{e_i} \mathsf{H}^T(\mathbf{x}) \cdot s_0(\mathbf{x})\, d^D\mathbf{x} \\ &= \int_{e_c} \mathsf{H}^T(\boldsymbol{\xi}) \cdot s_0(\boldsymbol{\xi}) \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}\right)\right]\right| d^D\boldsymbol{\xi} \\ &\approx \sum_{q=1}^{Q} \omega_q \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right| \cdot \mathsf{H}_c^T(\boldsymbol{\xi}_q) \cdot s_0(\boldsymbol{\xi}_q)\end{aligned}$$

y

$$\begin{aligned}\mathbf{b}_i^{(\Gamma_N)} &= \int_{e_i^{(D-1)}} \mathsf{H}^T(\mathbf{x}) \cdot p(\mathbf{x})\, d^{D-1}\mathbf{x} \\ &= \int_{e_c^{(D-1)}} \mathsf{H}^T(\boldsymbol{\xi}) \cdot p(\boldsymbol{\xi}) \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}\right)\right]\right| d^{D-1}\boldsymbol{\xi} \\ &\approx \sum_{q=1}^{Q} \omega_q \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right| \cdot \mathsf{H}^T(\boldsymbol{\xi}_q) \cdot p(\boldsymbol{\xi}_q)\end{aligned}$$

respectivamente.

Si existe una frontera Γ_V con condición de contorno de vacío, debemos agregar una contribución superficial a la matriz de rigidez

$$\begin{aligned}K_i^{(\Gamma_V)} &= \int_{e_i^{(D-1)}} \mathsf{H}^T(\mathbf{x}) \cdot \frac{1}{2} \cdot \mathsf{H}(\mathbf{x})\, d^{D-1}\mathbf{x} \\ &= \int_{e_c^{(D-1)}} \mathsf{H}^T(\boldsymbol{\xi}) \cdot \frac{1}{2} \cdot \mathsf{H}(\boldsymbol{\xi}) \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}\right)\right]\right| d^{D-1}\boldsymbol{\xi} \\ &\approx \sum_{q=1}^{Q} \omega_q \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right| \cdot \mathsf{H}^T(\boldsymbol{\xi}_q) \cdot \frac{1}{2} \cdot \mathsf{H}(\boldsymbol{\xi}_q)\end{aligned}$$

Observación. Para problemas de criticidad, el término de fisiones F_i no se suma a la matriz de rigidez K_i sino que forma parte de la matriz de masa M_i de forma tal de poder escribir las ecuaciones discretizadas como un problema de autovalores

$$\mathsf{K} \cdot \boldsymbol{\phi} = \frac{1}{k_{\text{eff}}} \cdot \mathsf{M} \cdot \boldsymbol{\phi}$$

En la sección 3.5.3 discutimos este caso con más detalle.

3.4.2.2. Grupos arbitrarios de energía

Consideremos primeramente caso a dos grupos de energías $G = 2$. La formulación fuerte ahora no es una sino dos ecuaciones diferenciales en derivadas parciales acopladas entre sí a través de los términos de scattering y de fisión

$$\begin{cases} -\text{div}\left[D_1 \,\text{grad}\left(\phi_1\right)\right] + \Sigma_{t1}\phi_1 - \Sigma_{s_0 1\to 1}\phi_1 - \Sigma_{s_0 2\to 1}\phi_2 - \chi_1\left[\nu\Sigma_{f1}\phi_1 + \nu\Sigma_{f2}\phi_2\right] = s_{0,1} \\ -\text{div}\left[D_2 \,\text{grad}\left(\phi_2\right)\right] + \Sigma_{t2}\phi_2 - \Sigma_{s_0 1\to 2}\phi_1 - \Sigma_{s_0 2\to 2}\phi_2 - \chi_2\left[\nu\Sigma_{f1}\phi_1 + \nu\Sigma_{f2}\phi_2\right] = s_{0,2} \end{cases}$$

que podemos escribir en una única ecuación vectorial como

$$\begin{bmatrix} -\text{div}\left[D_1(\mathbf{x}) \,\text{grad}\left(\phi_1\right)\right] \\ -\text{div}\left[D_2(\mathbf{x}) \,\text{grad}\left(\phi_2\right)\right] \end{bmatrix} + \begin{bmatrix} \Sigma_{t1}(\mathbf{x}) - \Sigma_{s_0 1\to 1}(\mathbf{x}) & -\Sigma_{s_0 2\to 1}(\mathbf{x}) \\ -\Sigma_{s_0 1\to 2}(\mathbf{x}) & \Sigma_{t2}(\mathbf{x}) - \Sigma_{s_0 2\to 2}(\mathbf{x}) \end{bmatrix} \cdot \begin{bmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \end{bmatrix} - \begin{bmatrix} \chi_1 \cdot \nu\Sigma_{f1}(\mathbf{x}) & \chi_1 \cdot \nu\Sigma_{f2}(\mathbf{x}) \\ \chi_2 \cdot \nu\Sigma_{f1}(\mathbf{x}) & \chi_2 \cdot \nu\Sigma_{f2}(\mathbf{x}) \end{bmatrix} \cdot \begin{bmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} s_{0,1}(\mathbf{x}) \\ ´s_{0,2}(\mathbf{x}) \end{bmatrix}$$

Introduciendo la matriz $\mathsf{R} \in \mathbb{R}^{2\times 2}$ de remociones

$$\mathsf{R}(\mathbf{x}) = \begin{bmatrix} \Sigma_{t1}(\mathbf{x}) & 0 \\ 0 & \Sigma_{t2}(\mathbf{x}) \end{bmatrix} - \begin{bmatrix} \Sigma_{s_0 1\to 1}(\mathbf{x}) & \Sigma_{s_0 2\to 1}(\mathbf{x}) \\ \Sigma_{s_0 1\to 2}(\mathbf{x}) & \Sigma_{s_0 2\to 2}(\mathbf{x}) \end{bmatrix}$$

y la matriz $\mathsf{X} \in \mathbb{R}^{2\times 2}$ de nu-fisiones

$$\mathsf{X}(\mathbf{x}) = \begin{bmatrix} \chi_1 \cdot \nu\Sigma_{f1}(\mathbf{x}) & \chi_1 \cdot \nu\Sigma_{f2}(\mathbf{x}) \\ \chi_2 \cdot \nu\Sigma_{f1}(\mathbf{x}) & \chi_2 \cdot \nu\Sigma_{f2}(\mathbf{x}) \end{bmatrix}$$

la formulación fuerte para dos grupos de energía $G = 2$ queda

$$\begin{bmatrix} -\text{div}\left[D_1(\mathbf{x}) \,\text{grad}\left(\phi_1\right)\right] \\ -\text{div}\left[D_2(\mathbf{x}) \,\text{grad}\left(\phi_2\right)\right] \end{bmatrix} + \mathsf{R}(\mathbf{x}) \cdot \begin{bmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \end{bmatrix} - \mathsf{X}(\mathbf{x}) \cdot \begin{bmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} s_{0,1}(\mathbf{x}) \\ s_{0,2}(\mathbf{x}) \end{bmatrix}$$

más las condiciones de contorno discutidas en la sección 2.4.5.

Para encontrar la formulación débil multiplicamos cada una de las dos ecuaciones por funciones de prueba (¿flujos escalares virtuales?) $v_1(\mathbf{x}) \in V$ y $v_2(\mathbf{x}) \in V$ respectivamente, las sumamos, integramos en el dominio $U \in \mathbb{R}^D$ y aplicamos la fórmula de Green al término de la divergencia:

$$\begin{aligned} \int_U \begin{bmatrix} \nabla v_1 & \nabla v_2 \end{bmatrix} \cdot \begin{bmatrix} D_1(\mathbf{x}) & 0 \\ 0 & D_2(\mathbf{x}) \end{bmatrix} \cdot \begin{bmatrix} \nabla\phi_1 \\ \nabla\phi_2 \end{bmatrix} d^D\mathbf{x} + \int_U \begin{bmatrix} v_1 & v_2 \end{bmatrix} \cdot \left[\mathsf{R}(\mathbf{x}) - \mathsf{X}(\mathbf{x})\right] \cdot \begin{bmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \end{bmatrix} d^D\mathbf{x} \\ = \int_U \begin{bmatrix} v_1 & v_2 \end{bmatrix} \cdot \begin{bmatrix} s_{0,1}(\mathbf{x}) \\ s_{0,2}(\mathbf{x}) \end{bmatrix} d^D\mathbf{x} + \int_{\Gamma_D} \begin{bmatrix} v_1 & v_2 \end{bmatrix} \cdot \begin{bmatrix} p_1(\mathbf{x}) \\ p_2(\mathbf{x}) \end{bmatrix} d^{D-1}\mathbf{x} \end{aligned}$$

Ahora tratamos de encontrar la aproximación de Galerkin. Para ello, definimos un vector $\mathbf{v} \in \mathbb{R}^{GJ}$ con los valores nodales de $v_1(\mathbf{x})$ para $g = 1$ y $v_2(\mathbf{x})$ para $g = 2$ intercalados

$$\mathbf{v} = \begin{bmatrix} v_1(\mathbf{x}_1) \\ v_2(\mathbf{x}_1) \\ v_1(\mathbf{x}_2) \\ v_2(\mathbf{x}_2) \\ \vdots \\ v_1(\mathbf{x}_J) \\ v_2(\mathbf{x}_J) \end{bmatrix}$$

Entonces

$$\begin{bmatrix} v_1(\mathbf{x}) \\ v_2(\mathbf{x}) \end{bmatrix} = \mathsf{H}_2(\mathbf{x}) \cdot \mathbf{v}$$

donde llamamos H_2 a la matriz de funciones de forma para $G = 2$

$$\mathsf{H}_2(\mathbf{x}) = \begin{bmatrix} h_1(\mathbf{x}) & 0 & h_2(\mathbf{x}) & 0 & \cdots & h_{J_i}(\mathbf{x}) & 0 \\ 0 & h_1(\mathbf{x}) & 0 & h_2(\mathbf{x}) & \cdots & 0 & h_{J_i}(\mathbf{x}) \end{bmatrix} \quad \in \mathbb{R}^{2\times 2J}$$

como un caso particular de la matriz de funciones de forma H_G para G grupos de energía. De la misma manera, si

$$\boldsymbol{\phi} = \begin{bmatrix} \phi_1(\mathbf{x}_1) \\ \phi_2(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) \\ \phi_2(\mathbf{x}_2) \\ \vdots \\ \phi_1(\mathbf{x}_J) \\ \phi_2(\mathbf{x}_J) \end{bmatrix}$$

entonces

$$\begin{bmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \end{bmatrix} = \mathsf{H}_2(\mathbf{x}) \cdot \boldsymbol{\phi}$$

y el término de remociones menos fisiones queda

$$\int_U \begin{bmatrix} v_1 & v_2 \end{bmatrix} \cdot \Big[\mathsf{R}(\mathbf{x}) - \mathsf{X}(\mathbf{x})\Big] \cdot \begin{bmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \end{bmatrix} d^D\mathbf{x} = \mathbf{v}^T \cdot \left[\int_U \mathsf{H}_2^T(\mathbf{x}) \cdot \Big[\mathsf{R}(\mathbf{x}) - \mathsf{X}(\mathbf{x})\Big] \cdot \mathsf{H}_2(\mathbf{x})\, d^D\mathbf{x}\right] \cdot \boldsymbol{\phi} \tag{3.57}$$

Análogamente, el término de fuentes volumétricas del miembro derecho queda

$$\int_U \begin{bmatrix} v_1 & v_2 \end{bmatrix} \cdot \begin{bmatrix} s_{0,1}(\mathbf{x}) \\ s_{0,2}(\mathbf{x}) \end{bmatrix} d^D\mathbf{x} = \mathbf{v}^T \cdot \left[\int_U \mathsf{H}_2^T(\mathbf{x}) \cdot \begin{bmatrix} s_{0,1}(\mathbf{x}) \\ s_{0,2}(\mathbf{x}) \end{bmatrix} d^D\mathbf{x}\right] \tag{3.58}$$

el de condiciones de contorno de Neumann

$$\int_{\Gamma_N} \begin{bmatrix} v_1 & v_2 \end{bmatrix} \cdot \begin{bmatrix} p_1(\mathbf{x}) \\ p_2(\mathbf{x}) \end{bmatrix} \, d^{D-1}\mathbf{x} = \mathbf{v}^T \cdot \left[\int_{\Gamma_N} \mathsf{H}_2^T(\mathbf{x}) \cdot \begin{bmatrix} p_1(\mathbf{x}) \\ p_2(\mathbf{x}) \end{bmatrix} \, d^{D-1}\mathbf{x} \right] \tag{3.59}$$

y de condiciones de vacío

$$\int_{\Gamma_V} \begin{bmatrix} v_1 & v_2 \end{bmatrix} \cdot \frac{1}{2} \cdot \begin{bmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \end{bmatrix} \, d^{D-1}\mathbf{x} = \mathbf{v}^T \cdot \left[\int_{\Gamma_V} \mathsf{H}_2^T(\mathbf{x}) \cdot \frac{1}{2} \cdot \mathsf{H}_2(\mathbf{x}) \, d^{D-1}\mathbf{x} \right] \tag{3.60}$$

Nos queda evaluar el término de pérdidas. Para ello, simplifiquemos primera la notación suponiendo un problema bi-dimensional $D = 2$. Por un lado notamos que

$$\begin{aligned} \begin{bmatrix} \nabla v_1 & \nabla v_2 \end{bmatrix} \cdot \begin{bmatrix} D_1(\mathbf{x}) & 0 \\ 0 & D_2(\mathbf{x}) \end{bmatrix} \cdot \begin{bmatrix} \nabla \phi_1 \\ \nabla \phi_2 \end{bmatrix} =& \frac{\partial v_1}{\partial x} \cdot D_1 \cdot \frac{\partial \phi_1}{\partial x} + \frac{\partial v_2}{\partial x} \cdot D_2 \cdot \frac{\partial \phi_2}{\partial x} \\ &+ \frac{\partial v_1}{\partial y} \cdot D_1 \cdot \frac{\partial \phi_1}{\partial y} + \frac{\partial v_2}{\partial y} \cdot D_2 \cdot \frac{\partial \phi_2}{\partial y} \\ =& \begin{bmatrix} \frac{\partial v_1}{\partial x} & \frac{\partial v_2}{\partial x} & \frac{\partial v_1}{\partial y} & \frac{\partial v_2}{\partial y} \end{bmatrix} \begin{bmatrix} D_1 & 0 & 0 & 0 \\ 0 & D_2 & 0 & 0 \\ 0 & 0 & D_1 & 0 \\ 0 & 0 & 0 & D_2 \end{bmatrix} \begin{bmatrix} \frac{\partial \phi_1}{\partial x} \\ \frac{\partial \phi_2}{\partial x} \\ \frac{\partial \phi_1}{\partial y} \\ \frac{\partial \phi_2}{\partial y} \end{bmatrix} \end{aligned}$$

y por el otro que

$$\begin{bmatrix} \frac{\partial v_1}{\partial x} \\ \frac{\partial v_2}{\partial x} \\ \frac{\partial v_1}{\partial y} \\ \frac{\partial v_2}{\partial y} \end{bmatrix} = \mathsf{B}_2(\mathbf{x}) \cdot \mathbf{v} \qquad \begin{bmatrix} \frac{\partial \phi_1}{\partial x} \\ \frac{\partial \phi_2}{\partial x} \\ \frac{\partial \phi_1}{\partial y} \\ \frac{\partial \phi_2}{\partial y} \end{bmatrix} = \mathsf{B}_2(\mathbf{x}) \cdot \boldsymbol{\phi}$$

con

$$\mathsf{B}_2(\mathbf{x}) = \begin{bmatrix} \frac{\partial h_1}{\partial x} & 0 & \frac{\partial h_2}{\partial x} & 0 & \cdots & \frac{\partial h_j}{\partial x} & 0 \\ 0 & \frac{\partial h_1}{\partial x} & 0 & \frac{\partial h_2}{\partial x} & \cdots & 0 & \frac{\partial h_j}{\partial x} \\ \frac{\partial h_1}{\partial y} & 0 & \frac{\partial h_2}{\partial y} & 0 & \cdots & \frac{\partial h_j}{\partial y} & 0 \\ 0 & \frac{\partial h_1}{\partial y} & 0 & \frac{\partial h_2}{\partial y} & \cdots & 0 & \frac{\partial h_j}{\partial y} \\ \frac{\partial h_1}{\partial z} & 0 & \frac{\partial h_2}{\partial z} & 0 & \cdots & \frac{\partial h_j}{\partial z} & 0 \\ 0 & \frac{\partial h_1}{\partial z} & 0 & \frac{\partial h_2}{\partial z} & \cdots & 0 & \frac{\partial h_j}{\partial z} \end{bmatrix} \in \mathbb{R}^{2D \times 2J}$$

entonces

$$\int_U \begin{bmatrix}\nabla v_1 & \nabla v_2\end{bmatrix} \cdot \begin{bmatrix}D_1(\mathbf{x}) & 0\\ 0 & D_2(\mathbf{x})\end{bmatrix} \cdot \begin{bmatrix}\nabla \phi_1\\ \nabla \phi_2\end{bmatrix} d^D\mathbf{x} = \mathbf{v}^T \cdot \left[\int_U \mathsf{B}_2^T(\mathbf{x}) \cdot \mathsf{D}'(\mathbf{x}) \cdot \mathsf{B}_2(\mathbf{x})\, d^D\mathbf{x}\right] \cdot \boldsymbol{\phi} \quad (3.61)$$

siendo la matriz diagonal $\mathsf{D} \in \mathbb{R}^{2\times 2}$

$$\mathsf{D}(\mathbf{x}) = \begin{bmatrix}D_1(\mathbf{x}) & 0\\ 0 & D_2(\mathbf{x})\end{bmatrix}$$

y construyendo la matriz bloque-diagonal D_D según la dimensión D del dominio U

$$\mathsf{D}_1(\mathbf{x}) = \mathsf{D}(\mathbf{x}) = \begin{bmatrix}D_1(\mathbf{x}) & 0\\ 0 & D_2(\mathbf{x})\end{bmatrix}$$

$$\mathsf{D}_2(\mathbf{x}) = \begin{bmatrix}\mathsf{D}(\mathbf{x}) & \mathsf{0}\\ \mathsf{0} & \mathsf{D}(\mathbf{x})\end{bmatrix} = \begin{bmatrix}D_1(\mathbf{x}) & 0 & 0 & 0\\ 0 & D_2(\mathbf{x}) & 0 & 0\\ 0 & 0 & D_1(\mathbf{x}) & 0\\ 0 & 0 & 0 & D_2(\mathbf{x})\end{bmatrix}$$

$$\mathsf{D}_3(\mathbf{x}) = \begin{bmatrix}\mathsf{D}(\mathbf{x}) & \mathsf{0} & \mathsf{0}\\ \mathsf{0} & \mathsf{D}(\mathbf{x}) & \mathsf{0}\\ \mathsf{0} & \mathsf{0} & \mathsf{D}(\mathbf{x})\end{bmatrix} = \begin{bmatrix}D_1(\mathbf{x}) & 0 & 0 & 0 & 0 & 0\\ 0 & D_2(\mathbf{x}) & 0 & 0 & 0 & 0\\ 0 & 0 & D_1(\mathbf{x}) & 0 & 0 & 0\\ 0 & 0 & 0 & D_2(\mathbf{x}) & 0 & 0\\ 0 & 0 & 0 & 0 & D_1(\mathbf{x}) & 0\\ 0 & 0 & 0 & 0 & 0 & D_2(\mathbf{x})\end{bmatrix}$$

Juntando las ecuaciones

- 3.57 (remociones menos fisiones)
- 3.58 (fuentes independientes)
- 3.59 (condiciones de Neumann)
- 3.60 (condiciones de vacío)
- 3.61 (pérdidas)

el problema de Galerkin para difusión de neutrones a $G = 2$ grupos es encontrar $\boldsymbol{\phi} \in \mathbb{R}^{2J}$ tal que

$$\mathbf{v}^T \left[\int_U \left\{\mathsf{B}_2^T(\mathbf{x})\mathsf{D}'(\mathbf{x})\mathsf{B}_2(\mathbf{x}) + \mathsf{H}_2^T(\mathbf{x})\left[\mathsf{R}(\mathbf{x}) - \mathsf{F}(\mathbf{x})\right]\mathsf{H}_2(\mathbf{x})\right\} d^D\mathbf{x} + \int_{\Gamma_V} \mathsf{H}_2^T(\mathbf{x})\frac{1}{2}\mathsf{H}_2(\mathbf{x})\, d^{D-1}\mathbf{x}\right] \boldsymbol{\phi}$$
$$= \mathbf{v}^T \cdot \left[\int_U \mathsf{H}_2^T \cdot \begin{bmatrix}s_{0,1}(\mathbf{x})\\ s_{0,2}(\mathbf{x})\end{bmatrix} d^D\mathbf{x} + \int_{\Gamma_N} \mathsf{H}_2^T \cdot \begin{bmatrix}p_1(\mathbf{x})\\ p_2(\mathbf{x})\end{bmatrix} d^{D-1}\mathbf{x}\right]$$

para todo $\mathbf{v} \in \mathbb{R}^{2J}$.

Luego, para el caso general de G grupos de energía sobre un dominio de dimensión D, podemos escribir la matriz de rigidez elemental K_i como

$$\mathsf{K}_i = \sum_{q=1}^{Q} \omega_q \cdot \left| \det \left[\mathsf{J}_i \left(\boldsymbol{\xi}_q \right) \right] \right| \cdot \left[\mathsf{L}_i(\boldsymbol{\xi}_q) + \mathsf{A}_i(\boldsymbol{\xi}_q) - \mathsf{F}_i(\boldsymbol{\xi}_q) \right] \tag{3.62}$$

a partir de las matrices elementales de pérdidas, absorciones y fisiones de tamaño $GJ \times GJ$

$$\begin{aligned} \mathsf{L}_i &= \mathsf{B}_{Gi}^T(\boldsymbol{\xi}_q) \cdot \mathsf{D}_D(\boldsymbol{\xi}_q) \cdot \mathsf{B}_{Gi}(\boldsymbol{\xi}_q) \\ \mathsf{A}_i &= \mathsf{H}_{Gc}^T(\boldsymbol{\xi}_q) \cdot \mathsf{R}(\boldsymbol{\xi}_q) \cdot \mathsf{H}_{Gc}(\boldsymbol{\xi}_q) \\ \mathsf{F}_i &= \mathsf{H}_{Gc}^T(\boldsymbol{\xi}_q) \cdot \mathsf{X}(\boldsymbol{\xi}_q) \cdot \mathsf{H}_{Gc}(\boldsymbol{\xi}_q) \end{aligned} \tag{3.63}$$

donde tenemos ahora la matriz $\mathsf{H}_{Gc} \in \mathbb{R}^{G \times GJ}$ de funciones de forma canónica para G grados de libertad por nodo

$$\mathsf{H}_{Gc}(\boldsymbol{\xi}) = \begin{bmatrix} h_1(\boldsymbol{\xi}) & 0 & \cdots & 0 & h_2(\boldsymbol{\xi}) & 0 & \cdots & 0 & \cdots & h_{J_i}(\boldsymbol{\xi}) & 0 & \cdots & 0 \\ 0 & h_1(\boldsymbol{\xi}) & \cdots & 0 & 0 & h_2(\boldsymbol{\xi}) & \cdots & 0 & \cdots & 0 & h_{J_i}(\boldsymbol{\xi}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & h_1(\boldsymbol{\xi}) & 0 & 0 & \cdots & h_2(\boldsymbol{\xi}) & \cdots & 0 & 0 & \cdots & h_{J_i}(\boldsymbol{\xi}) \end{bmatrix}$$

y la matriz de derivadas reales $\mathsf{B}_{Gi} \in \mathbb{R}^{GD \times GJ_i}$

$$\mathsf{B}_{Gi}(\boldsymbol{\xi}) = \begin{bmatrix} \frac{\partial h_1}{\partial x} & 0 & \cdots & 0 & \frac{\partial h_2}{\partial x} & 0 & \cdots & 0 & \cdots & \frac{\partial h_{J_i}}{\partial x} & 0 & \cdots & 0 \\ 0 & \frac{\partial h_1}{\partial x} & \cdots & 0 & 0 & \frac{\partial h_2}{\partial x} & \cdots & 0 & \cdots & 0 & \frac{\partial h_{J_i}}{\partial x} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial h_1}{\partial x} & 0 & 0 & \cdots & \frac{\partial h_2}{\partial x} & \cdots & 0 & 0 & \cdots & \frac{\partial h_{J_i}}{\partial x} \\ \frac{\partial h_1}{\partial y} & 0 & \cdots & 0 & \frac{\partial h_2}{\partial y} & 0 & \cdots & 0 & \cdots & \frac{\partial h_{J_i}}{\partial y} & 0 & \cdots & 0 \\ 0 & \frac{\partial h_1}{\partial y} & \cdots & 0 & 0 & \frac{\partial h_2}{\partial y} & \cdots & 0 & \cdots & 0 & \frac{\partial h_{J_i}}{\partial y} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial h_1}{\partial y} & 0 & 0 & \cdots & \frac{\partial h_2}{\partial y} & \cdots & 0 & 0 & \cdots & \frac{\partial h_{J_i}}{\partial y} \\ \frac{\partial h_1}{\partial z} & 0 & \cdots & 0 & \frac{\partial h_2}{\partial z} & 0 & \cdots & 0 & \cdots & \frac{\partial h_{J_i}}{\partial z} & 0 & \cdots & 0 \\ 0 & \frac{\partial h_1}{\partial z} & \cdots & 0 & 0 & \frac{\partial h_2}{\partial z} & \cdots & 0 & \cdots & 0 & \frac{\partial h_{J_i}}{\partial z} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial h_1}{\partial z} & 0 & 0 & \cdots & \frac{\partial h_2}{\partial z} & \cdots & 0 & 0 & \cdots & \frac{\partial h_{J_i}}{\partial z} \end{bmatrix}$$

con las matrices de secciones eficaces macroscópicas de difusión, remoción y nu-fisión

$$\mathsf{D}(\mathbf{x}) = \begin{bmatrix} D_1(\mathbf{x}) & 0 & \cdots & 0 \\ 0 & D_2(\mathbf{x}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D_G(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^{G\times G}$$

$$\mathsf{D}_D(\mathbf{x}) = \begin{bmatrix} \mathsf{D}(\mathbf{x}) & \mathsf{0} & \cdots & \mathsf{0} \\ \mathsf{0} & \mathsf{D}(\mathbf{x}) & \cdots & \mathsf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathsf{0} & \mathsf{0} & \cdots & \mathsf{D}(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^{GD\times GD}$$

$$\mathsf{R} = \begin{bmatrix} \Sigma_{t1}(\mathbf{x}) & 0 & \cdots & 0 \\ 0 & \Sigma_{t2}(\mathbf{x}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Sigma_{tG}(\mathbf{x}) \end{bmatrix} - \begin{bmatrix} \Sigma_{s_0 1\to 1}(\mathbf{x}) & \Sigma_{s_0 2\to 1} & \cdots & \Sigma_{s_0 G\to 1} \\ \Sigma_{s_0 1\to 2}(\mathbf{x}) & \Sigma_{s_0 2\to 2} & \cdots & \Sigma_{s_0 G\to 2} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{s_0 1\to G}(\mathbf{x}) & \Sigma_{s_0 2\to G} & \cdots & \Sigma_{s_0 G\to G} \end{bmatrix} \in \mathbb{R}^{G\times G}$$

$$\mathsf{X} = \begin{bmatrix} \chi_1\cdot\nu\Sigma_{f1}(\mathbf{x}) & \chi_1\cdot\nu\Sigma_{f2}(\mathbf{x}) & \cdots & \chi_1\cdot\nu\Sigma_{fG}(\mathbf{x}) \\ \chi_2\cdot\nu\Sigma_{f1}(\mathbf{x}) & \chi_2\cdot\nu\Sigma_{f2}(\mathbf{x}) & \cdots & \chi_2\cdot\nu\Sigma_{fG}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \chi_G\cdot\nu\Sigma_{f1}(\mathbf{x}) & \chi_G\cdot\nu\Sigma_{f2}(\mathbf{x}) & \cdots & \chi_G\cdot\nu\Sigma_{fG}(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^{G\times G}$$

Observación. El operador bi-lineal $a([\phi_1 \ \dots \ \phi_G]^T, [v_1 \ \dots \ v_G]) : V^G \times V^G \mapsto \mathbb{R}$ para $G > 1$ es

$$\begin{aligned} a([\phi_1 \ \dots \ \phi_G]^T, [v_1 \ \dots \ v_G]^T) = &\int_U \text{grad}\Big[v(\mathbf{x})\Big] \cdot \mathsf{D}'(\mathbf{x}) \cdot \text{grad}\Big[\phi(\mathbf{x})\Big]\, d^D\mathbf{x} \\ &+ \int_U v(\mathbf{x}) \cdot [\mathsf{R}(\mathbf{x}) - \mathsf{X}(\mathbf{x})] \cdot \phi(\mathbf{x})\, d^D\mathbf{x} \\ &+ \int_{\Gamma_V} v(\mathbf{x}) \cdot \frac{1}{2} \cdot \phi(\mathbf{x})\, d^{D-1}\mathbf{x} \\ = &\int_U \mathbf{v}^T \cdot \mathsf{B}^T(\mathbf{x}) \cdot \mathsf{D}'(\mathbf{x}) \cdot \mathsf{B}(\mathbf{x}) \cdot \mathbf{u}\ d^D\mathbf{x} \\ &+ \int_U \mathbf{v}^T \cdot \mathsf{H}^T(\mathbf{x}) \cdot [\mathsf{R}(\mathbf{x}) - \mathsf{X}(\mathbf{x})] \cdot \mathsf{H}(\mathbf{x}) \cdot \boldsymbol{\phi}\, d^D\mathbf{x} \\ &+ \int_{\Gamma_V} \mathbf{v}^T \cdot \mathsf{H}^T(\mathbf{x}) \cdot \frac{1}{2} \cdot \mathsf{H}(\mathbf{x}) \cdot \boldsymbol{\phi}\, d^{D-1}\mathbf{x} \\ = &\mathbf{v}^T \cdot \Bigg[\int_U \mathsf{B}^T(\mathbf{x}) \cdot \mathsf{D}'(\mathbf{x}) \cdot \mathsf{B}(\mathbf{x})\, d^D\mathbf{x} \\ &+ \int_U \mathsf{H}^T(\mathbf{x}) \cdot [\mathsf{R}(\mathbf{x}) - \mathsf{X}(\mathbf{x})] \cdot \mathsf{H}(\mathbf{x})\, d^D\mathbf{x} \\ &+ \int_{\Gamma_V} \mathsf{H}^T(\mathbf{x}) \cdot \frac{1}{2} \cdot \mathsf{H}(\mathbf{x})\, d^{D-1}\mathbf{x}\Bigg] \cdot \boldsymbol{\phi} \end{aligned} \tag{3.64}$$

Observación. El operador de la ecuación 3.64 es simétrico sólo si ambas matrices R y X también lo son.

Observación. Las matrices R y X no son simétricas para propiedades nucleares reales.

Observación. El operador de la ecuación 3.64 es coercitivo sólo si $k_{\text{eff}} < 1$.

Observación. Las matrices H_{Gc} y B_{Gi} que "saben" (decimos que son *G-aware*) cuántos grupos de energía tiene el problema se construyen a partir de las matrices del problema escalar H_c y B_i como

$$\mathsf{H}_{Gc}\Big(g, G\cdot(j-1)+g\Big) = H_c(1,j) = h_j(\boldsymbol{\xi})$$

$$\mathsf{B}_{Gi}\Big(G\cdot(d-1)+g, G\cdot(j-1)+g\Big) = B_c(d,j) = \frac{\partial h_j}{\partial \xi_d}$$

para $j = 1, \dots, J$, $g = 1, \dots, G$, y $d = 1, \dots, D$. El resto de los elementos son cero.

Observación. Para $G = 1$,

$$\mathsf{H}_{1c} = \mathsf{H}_c \quad \text{y} \quad \mathsf{B}_{1i} = \mathsf{B}_c$$

Observación. La forma de las matrices G-aware puede ser diferente para otros problemas vectoriales con el mismo número de grados de libertad. Por ejemplo, para el problema de elasticidad tridimensional basado en desplazamientos, $G = 3$ pero la matriz B_{3i} es (ver [9], tabla 6.6)

$$\mathsf{B}_{3i} = \begin{bmatrix} \frac{\partial h_1}{\partial x} & 0 & 0 & \frac{\partial h_2}{\partial x} & 0 & 0 & \cdots & \frac{\partial h_{J_i}}{\partial x} & 0 & 0 \\ 0 & \frac{\partial h_1}{\partial y} & 0 & 0 & \frac{\partial h_2}{\partial y} & 0 & \cdots & 0 & \frac{\partial h_{J_i}}{\partial y} & 0 \\ 0 & 0 & \frac{\partial h_1}{\partial z} & 0 & 0 & \frac{\partial h_2}{\partial z} & \cdots & 0 & 0 & \frac{\partial h_{J_i}}{\partial z} \\ \frac{\partial h_1}{\partial y} & \frac{\partial h_1}{\partial x} & 0 & \frac{\partial h_2}{\partial y} & \frac{\partial h_2}{\partial x} & 0 & \cdots & \frac{\partial h_{J_i}}{\partial y} & \frac{\partial h_{J_i}}{\partial x} & 0 \\ 0 & \frac{\partial h_1}{\partial z} & \frac{\partial h_1}{\partial y} & 0 & \frac{\partial h_2}{\partial z} & \frac{\partial h_2}{\partial y} & \cdots & 0 & \frac{\partial h_{J_i}}{\partial z} & \frac{\partial h_{J_i}}{\partial y} \\ \frac{\partial h_1}{\partial z} & 0 & \frac{\partial h_1}{\partial x} & \frac{\partial h_2}{\partial z} & 0 & \frac{\partial h_2}{\partial x} & \cdots & \frac{\partial h_{J_i}}{\partial z} & 0 & \frac{\partial h_{J_i}}{\partial x} \end{bmatrix}$$

ya que la matriz equivalente a la D_G del problema de difusión es de tamaño 6×6 con tres filas para las tensiones normales y tres filas para los esfuerzos de corte en una notación de Voigt.

3.4.3. Ordenadas discretas multigrupo

Siguiendo el mismo razonamiento que en la sección anterior, para el caso de ordenadas discretas multigrupo tenemos que derivar un operador $a(\psi, v) : V^{MG} \times V^{MG} \mapsto \mathbb{R}$ y un funcional $\mathcal{B}(v) : V^{MG} \mapsto \mathbb{R}$ de forma tal de re-escribir la formulación fuerte dada por la ecuación 3.16 en una formulación débil como

$$\text{encontrar } [\psi_{11}(\mathbf{x}) \ \cdots \ \psi_{MG}(\mathbf{x})]^T \in V^{MG} \text{ tal que}$$
$$a\Big([\psi_{11} \ \cdots \ \psi_{MG}]^T, [v_{11} \ \cdots \ v_{MG}]^T\Big) = \mathcal{B}\Big([v_{11} \ \cdots \ v_{MG}]^T)\Big) \quad \forall [v_{11}(\mathbf{x}) \ \cdots \ v_{MG}(\mathbf{x})]^T \in V^{MG}$$

Para ello, partimos de la la ecuación 3.16

$$\underbrace{\hat{\mathbf{\Omega}}_m \cdot \text{grad}\left[\psi_{mg}(\mathbf{x})\right]}_{\text{advección}} + \underbrace{\Sigma_{tg}(\mathbf{x}) \cdot \psi_{mg}(\mathbf{x})}_{\text{absorciones}} = \underbrace{\sum_{g'=1}^{G} \Sigma_{s_0 g' \to g}(\mathbf{x}) \sum_{m'=1} w_{m'} \psi_{m'g'}(\mathbf{x}) +}_{\text{scattering isotrópico}}$$

$$\underbrace{3 \sum_{g'=1}^{G} \Sigma_{s_1 g' \to g}(\mathbf{x}) \sum_{m'=1} w_{m'} \left(\hat{\mathbf{\Omega}}_m \cdot \hat{\mathbf{\Omega}}_{m'}\right) \psi_{m'g'}(\mathbf{x})}_{\text{scattering anisótropo}} + \underbrace{\chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}(\mathbf{x}) \sum_{m'=1} w_{m'} \psi_{m'g'}(\mathbf{x})}_{\text{fisiones}} + \underbrace{s_{mg}(\mathbf{x})}_{\text{fuentes}}$$

y tal como hicimos con la ecuación de difusión, multiplicamos cada término por $v_{mg}(\mathbf{x})$, integramos sobre el dominio $U \in \mathbb{R}^D$, pasamos los términos de scattering y fisiones al miembro izquierdo y los sumamos. Podemos escribir los términos de absorciones, scattering y fisiones en forma similar al caso de difusión como

$$\int_U \begin{bmatrix} v_{11}(\mathbf{x}) & \cdots & v_{MG}(\mathbf{x}) \end{bmatrix} \cdot \Big(\mathsf{R}(\mathbf{x}) - \mathsf{F}(\mathbf{x})\Big) \cdot \begin{bmatrix} \psi_{11}(\mathbf{x}) \\ \vdots \\ \psi_{MG}(\mathbf{x}) \end{bmatrix} d^D\mathbf{x}$$

con las matrices cuadradas de secciones eficaces de remoción y ν-fisiones de tamaño $MG \times MG$

$$\mathsf{R}(\mathbf{x}) = \begin{bmatrix} \Sigma_{t1} - w_1 \cdot \Sigma_{s_0\ 1\to1} & -w_1 \cdot \Sigma_{s_0\ 1\to2} & \cdots & -w_1 \cdot \Sigma_{s_0\ 1\to G} & -w_1 \cdot \Sigma_{s_0\ 2\to1} & \cdots \end{bmatrix}$$

$$\mathsf{X}(\mathbf{x}) = \begin{bmatrix} w_1\chi_1\nu\Sigma_{f1} & w_1\chi_1\nu\Sigma_{f2} & w_1\chi_1\nu\Sigma_{f1} & \cdots & w_1\chi_1\nu\Sigma_{fG} & w_1\chi_2\nu\Sigma_{f1} & \cdots \end{bmatrix}$$

Pero ahora el término de advección es de primer orden

$$\int_U \begin{bmatrix} v_{11}(\mathbf{x}) & \cdots & v_{mg}(\mathbf{x}) & \cdots & v_{MG}(\mathbf{x}) \end{bmatrix} \cdot \begin{bmatrix} \hat{\mathbf{\Omega}}_1 \cdot \nabla\psi_{11}(\mathbf{x}) \\ \vdots \\ \hat{\Omega}_{mx} \cdot \frac{\partial\psi_{mg}}{\partial x} + \hat{\Omega}_{my} \cdot \frac{\partial\psi_{mg}}{\partial y} + \hat{\Omega}_{mz} \cdot \frac{\partial\psi_{mg}}{\partial z} \\ \vdots \\ \hat{\mathbf{\Omega}}_M \cdot \nabla\psi_{MG}(\mathbf{x}) \end{bmatrix} d^D\mathbf{x}$$

Re-escribimos el vector $\Omega_m \cdot \nabla\psi_{mg}$ como el producto de una matriz constante con los cosenos directores D de tamaño $MG \times MGD$ y un vector $\psi' \in \mathbb{R}^{MG}$ de derivadas parciales de los MG flujos angulares con respecto a las D coordenadas

$$\begin{bmatrix} \hat{\Omega}_{1x} & 0 & \cdots & 0 & \cdots & 0 & \hat{\Omega}_{1y} & \cdots & 0 \\ 0 & \hat{\Omega}_{1x} & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \hat{\Omega}_{mx} & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & \cdots & \hat{\Omega}_{Mx} & 0 & \cdots & \hat{\Omega}_{Mz} \end{bmatrix} \cdot \begin{bmatrix} \partial\psi_{11}/\partial x \\ \partial\psi_{12}/\partial x \\ \vdots \\ \partial\psi_{mg}/\partial x \\ \vdots \\ \partial\psi_{MG}/\partial x \\ \partial\psi_{11}/\partial y \\ \vdots \\ \partial\psi_{MG}/\partial z \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{\Omega}}_1 \cdot \nabla\psi_{11}(\mathbf{x}) \\ \hat{\mathbf{\Omega}}_1 \cdot \nabla\psi_{12}(\mathbf{x}) \\ \vdots \\ \hat{\mathbf{\Omega}}_m \cdot \nabla\psi_{mg}(\mathbf{x}) \\ \vdots \\ \hat{\mathbf{\Omega}}_M \cdot \nabla\psi_{MG}(\mathbf{x}) \end{bmatrix}$$

El operador $a([\psi_{11} \dots \psi_{MG}]^T, [v_{11} \dots v_{MG}]^T) : V^{MG} \times V^{MG} \mapsto \mathbb{R}$ que buscamos es

$$\begin{aligned} a([\psi_{11} \dots \psi_{MG}]^T, [v_{11} \dots v_{MG}]^T) = & \int_U [v_{11}(\mathbf{x}) \quad \cdots \quad v_{MG}(\mathbf{x})] \cdot \mathsf{D} \cdot \boldsymbol{\psi}'(\mathbf{x}) d^D\mathbf{x} \\ & + \int_U [v_{11}(\mathbf{x}) \quad \cdots \quad v_{MG}(\mathbf{x})] \left(\mathsf{R}(\mathbf{x}) - \mathsf{F}(\mathbf{x}) \right) \begin{bmatrix} \psi_{11}(\mathbf{x}) \\ \vdots \\ \psi_{MG}(\mathbf{x}) \end{bmatrix} d^D\mathbf{x} \end{aligned}$$

Observación. El operador $a([\psi_{11} \dots \psi_{MG}]^T, [v_{11} \dots v_{MG}]^T)$ para ordenadas discretas no es simétrico y la mayoría de las veces tampoco es coercitivo.

Como en la formulación S_N las condiciones de contorno son sólo de Dirichlet, la única contribución al vector $\mathbf{b}$ proviene del término de fuentes independientes. Entonces el funcional $\mathcal{B}(v) : V^{MG} \mapsto \mathbb{R}$ es directamente

$$\mathcal{B}([v_{11} \dots v_{MG}]^T) = \int_U [v_{11}(\mathbf{x}) \quad \cdots \quad v_{MG}(\mathbf{x})] \cdot \begin{bmatrix} s_{11}(\mathbf{x}) \\ \vdots \\ s_{MG}(\mathbf{x}) \end{bmatrix} d^D\mathbf{x}$$

En principio, estaríamos en condiciones de discretizar la variable espacial $\mathbf{x}$ con las matrices H_{MGc} y B_{MG} tal como hemos hecho en la sección 3.4.2.2 para la ecuación de difusión multigrupo, con la salvedad de que ahora hay MG grados de libertad por nodo espacial. Pero el hecho de que el operador no sea coercitivo hace que el método numérico basado en la aproximación de Galerkin no sea estable y por lo tanto no converja. Una forma de recuperar la coercitividad del operador a y poder obtener una solución numérica al problema de ordenadas discretas formulado con un esquema de elementos finitos sobre la variable espacial $\mathbf{x}$ es resolver un problema de Petrov-Galerkin en el cual cada una de las funciones de prueba v_{mg} vive en un espacio vectorial V'_N diferente al espacio vectorial V_N donde viven las incógnitas ψ_{mg} para alguna elección adecuada de V'_N.

Definición 3.26 (problema de Petrov-Galerkin). Sea V_N un sub-espacio de $V = H_0^1(U)$ y sea V_N' un sub-espacio de $V' = H_0'^1(U)$, ambos de dimensión finita N Llamamos *problema de Petrov-Galerkin* a

$$\text{encontrar } u_N \in V_N : \quad a(u_N, v_N) = \mathcal{B}(v_N) \quad \forall v_N \in V_N'$$

Para encontrar la formulación de Petrov-Galerkin del problema de transporte multigrupo de neutrones por ordenadas discretas, comenzamos aproximando las incógnitas de la misma manera que para el problema de Galerkin

$$\begin{bmatrix} \psi_{11}(\mathbf{x}) \\ \vdots \\ \psi_{MG}(\mathbf{x}) \end{bmatrix} = \mathsf{H}_{MGc}(\mathbf{x}) \cdot \boldsymbol{\psi}$$

donde $\boldsymbol{\psi} \in \mathbb{R}^{MG}$ es un vector que contiene los valores nodales de los flujos angulares y H_{MGc} es la matriz canónica de funciones de forma MG-aware. Pero como las funciones de prueba viven en otro espacio vectorial V_N', ahora

$$\begin{bmatrix} v_{11}(\mathbf{x}) \\ \vdots \\ v_{MG}(\mathbf{x}) \end{bmatrix} = \mathsf{P}_{MGc}(\mathbf{x}) \cdot \mathbf{v}$$

para un vector $\mathbf{v} \in \mathbb{R}^{MG}$ con los valores nodales de las funciones de prueba pero para una matriz de Petrov P_{MGc} con las funciones de forma que generan[23] el espacio V_N'. Entonces la formulación débil discretizada queda

$$\mathbf{v}^T \cdot \left[\int_U \left(\mathsf{P}_{MGc}^T(\mathbf{x}) \cdot \mathsf{D} \cdot \mathsf{H}_{MGc}(\mathbf{x}) + \mathsf{H}_{MGc}^T \cdot \left(\mathsf{R}(\mathbf{x}) - \mathsf{X}(\mathbf{x}) \right) \cdot \mathsf{H}_{MGc}(\mathbf{x}) \right) d^D\mathbf{x} \right] \cdot \boldsymbol{\psi} =$$
$$\mathbf{v}^T \cdot \left[\int_U \left(\mathsf{P}_{MGc}^T(\mathbf{x}) \cdot \mathbf{s}(\mathbf{x}) \right) d^D\mathbf{x} \right]$$

lo que implica

$$\mathsf{K} \cdot \boldsymbol{\psi} = \mathbf{b}$$

De la misma manera que para difusión, la matriz de rigidez elemental de tamaño $MGJ \times MGJ$ evaluada en el q-ésimo punto de Gauss $\boldsymbol{\xi}_q$ es

$$\mathsf{K}_i = \sum_{q=1}^{Q} \omega_q \cdot \left| \det \left[\mathsf{J}_i \left(\boldsymbol{\xi}_q \right) \right] \right| \cdot \left[\mathsf{L}_i(\boldsymbol{\xi}_q) + \mathsf{A}_i(\boldsymbol{\xi}_q) - \mathsf{F}_i(\boldsymbol{\xi}_q) \right] \tag{3.65}$$

a partir de las matrices elementales de pérdidas, absorciones y fisiones de tamaño $MGJ \times MGJ$

[23] Del inglés *span*.

$$\begin{aligned}\mathsf{L}_i &= \mathsf{P}^T_{MGc}(\boldsymbol{\xi}_q) \cdot \mathsf{D} \cdot \mathsf{H}_{MGc}(\boldsymbol{\xi}_q)\\ \mathsf{A}_i &= \mathsf{P}^T_{MGc}(\boldsymbol{\xi}_q)^T \cdot \mathsf{R}(\boldsymbol{\xi}_q) \cdot \mathsf{H}_{MGc}(\boldsymbol{\xi}_q)\\ \mathsf{F}_i &= \mathsf{P}^T_{MGc}(\boldsymbol{\xi}_q)^T \cdot \mathsf{X}(\boldsymbol{\xi}_q) \cdot \mathsf{H}_{MGc}(\boldsymbol{\xi}_q)\end{aligned}$$

Observación. En FeenoX, la matriz de estabilización SUPG P_{MGc} se calcula de la siguiente manera:

```
int MG = neutron_sn.directions * neutron_sn.groups;
double tau = feenox_var_value(neutron_sn.sn_alpha) * e->type->size(e);

gsl_matrix *H_G = feenox_fem_compute_H_Gc_at_gauss(e, q, feenox.pde.mesh->integration);
gsl_matrix *B = feenox_fem_compute_B_at_gauss(e, q, feenox.pde.mesh->integration);
feenox_call(gsl_matrix_memcpy(neutron_sn.P, H_G));
for (unsigned int j = 0; j < neutron_sn.n_nodes; j++) {
  int MGj = MG*j;
  for (unsigned int m = 0; m < neutron_sn.directions; m++) {
    for (unsigned int d = 0; d < feenox.pde.dim; d++) {
      double value = tau * neutron_sn.Omega[m][d] * gsl_matrix_get(B, d, j);
      for (unsigned int g = 0; g < neutron_sn.groups; g++) {
        int diag = sn_dof_index(m,g);
        gsl_matrix_add_to_element(neutron_sn.P, diag, MGj + diag, value);
      }
    }
  }
}
```

3.5. Problemas de estado estacionario

Si bien en el capítulo 2 hemos mantenido por completitud la dependencia temporal explícitamente en los flujos y corrientes, en esta tesis resolvemos solamente problemas de estado estacionario. Tal como hemos hecho en este capítulo, al eliminar el término de la derivada con respecto al tiempo, las propiedades matemáticas de las ecuaciones cambian y por lo tanto debemos resolverlas en forma diferente según tengamos alguno de los siguientes tres casos:

1. Medio no multiplicativo con fuentes independientes,
2. Medio multiplicativo con fuentes independientes, y
3. Medio multiplicativo sin fuentes independientes.

3.5.1. Medio no multiplicativo con fuentes independientes

Un medio no multiplicativo es aquel que no contiene núcleos capaces de fisionar. Cada neutrón que encontremos en el medio debe entonces provenir de una fuente externa s. Para estudiar este tipo de problemas, además de eliminar la derivada temporal y la dependencia con el tiempo, tenemos que hacer cero el término de fisión, por lo que $\nu\Sigma_g = 0\ \forall g$. Luego la ecuación de difusión queda

$$-\text{div}\Big[D(\mathbf{x},E)\cdot\text{grad}\left[\phi(\mathbf{x},E)\right]\Big]+\Sigma_t(\mathbf{x},E)\cdot\phi(\mathbf{x},E)=\int_0^\infty\Sigma_{s_0}(\mathbf{x},E'\to E)\cdot\phi(\mathbf{x},E')\,dE'+s_0(\mathbf{x},E) \tag{3.66}$$

y la de transporte

$$\begin{aligned}\hat{\mathbf{\Omega}}\cdot\text{grad}\left[\psi(\mathbf{x},\hat{\mathbf{\Omega}},E)\right]+\Sigma_t(\mathbf{x},E)\cdot\psi(\mathbf{x},\hat{\mathbf{\Omega}},E)=\\ \frac{1}{4\pi}\cdot\int_0^\infty\Sigma_{s_0}(\mathbf{x},E'\to E)\cdot\int_{4\pi}\psi(\mathbf{x},\hat{\mathbf{\Omega}}',E')\,d\hat{\mathbf{\Omega}}'\,dE'+\\ \frac{3\cdot\hat{\mathbf{\Omega}}}{4\pi}\cdot\int_0^\infty\Sigma_{s_1}(\mathbf{x},E'\to E)\cdot\int_{4\pi}\psi(\mathbf{x},\hat{\mathbf{\Omega}}',E')\cdot\hat{\mathbf{\Omega}}'\,d\hat{\mathbf{\Omega}}'\,dE'+s(\mathbf{x},\hat{\mathbf{\Omega}},E)\end{aligned} \tag{3.67}$$

Para que la solución sea no trivial, el vector $\mathbf{b}$ del miembro derecho debe ser no nulo lo que implica que

a. la fuente no se debe anular idénticamente en el dominio, y/o
b. las condiciones de contorno deben ser no homogéneas.

Si las secciones eficaces (incluyendo el coeficiente de difusión) dependen explícitamente de la posición $\mathbf{x}$ pero no dependen del flujo ψ o ϕ, entonces tanto la ecuación 3.66 como la 3.67 son lineales. Entonces tenemos un sistema de ecuaciones lineales que podemos escribir en forma matricial como

$$\mathsf{A}_N(\Sigma_N)\cdot\boldsymbol{\varphi}_N=\mathbf{b}_N(\Sigma_N) \tag{3.68}$$

donde

- $\boldsymbol{\varphi}_N$ es un vector de tamaño $N\in\mathbb{N}$ que contiene los valores nodales de la incógnita (flujo angular ψ en transporte y flujo escalar ϕ en difusión) asociada a cada uno de los grados de libertad del problema discretizado (cantidad de incógnitas espaciales, grupos de energía y/o direcciones),
- $\mathsf{A}_N(\Sigma_N)\in\mathbb{R}^{N\times N}$ es una matriz rala[24] cuadrada que contiene información sobre
  a. las secciones eficaces macroscópicas, es decir los coeficientes de la ecuacion que estamos resolviendo, y
  b. la discretización de los operadores diferenciales e integrales,
- $\mathbf{b}_N(\Sigma_N)\in\mathbb{R}^N$ es un vector que contiene
  a. la versión discretizada de la fuente independiente s, y/o
  b. las condiciones de contorno no homogéneas
- N es el tamaño del problema discretizado, que es el producto de
  a. la cantidad J de incógnitas espaciales (cantidad de nodos en elementos finitos y cantidad de celdas en volúmenes finitos),

[24] Del inglés *sparse*.

b. la cantidad G de grupos de energía, y
c. la cantidad M de direcciones discretas (sólo para el método de ordenadas discretas).

El vector $\boldsymbol{\varphi}_N \in \mathbb{R}^N$ es la incógnita, que luego de resolver el sistema permitirá evaluar en forma aproximada (en el sentido de la sección 3.1) la función ψ ó ϕ en función de $\mathbf{x}$, E y eventualmente $\hat{\boldsymbol{\Omega}}$ para todo punto del espacio $\mathbf{x}$ dependiendo de la discretización espacial.

Observación. Las secciones 5.2, 5.4 y 5.9 contienen casos de este tipo de problemas.

Si las secciones eficaces dependen directa o indirectamente del flujo, por ejemplo a través de concentraciones de venenos o de la temperatura de los materiales (que a su vez puede depender de la potencia disipada, que depende del flujo neutrónico) entonces el problema es no lineal. En este caso, tenemos que volver a escribir la versión discretizada en forma genérica 3.1

$$\mathcal{F}_N(\boldsymbol{\varphi}_N, \Sigma_N) = 0 \tag{3.1}$$

para alguna función vectorial $\mathcal{F}_N : [\mathbb{R}^N \times \mathbb{R}^{N'}] \mapsto \mathbb{R}^N$.[25] La forma más eficiente de resolver estos problemas es utilizar variaciones del esquema de Newton [7], donde la incógnita $\boldsymbol{\varphi}_N$ se obtiene iterando a partir de una estimación inicial[26] $\boldsymbol{\varphi}_{N0}$

$$\boldsymbol{\varphi}_{Nk+1} = \boldsymbol{\varphi}_{Nk} - \mathsf{J}_N(\boldsymbol{\varphi}_{Nk}, \Sigma_{Nk})^{-1} \cdot \mathcal{F}_N(\boldsymbol{\varphi}_{Nk}, \Sigma_{Nk})$$

para los pasos $k = 0, 1, ...$, donde J_N es la matriz jacobiana de la función $\mathcal{F}_N$. Dado que la inversa de una matriz rala es densa, es prohibitivo evaluar (¡y almacenar!) explícitamente J_N^{-1}. En la práctica, la iteración de Newton se implementa mediante los siguientes dos pasos:

1. Resolver $\mathsf{J}(\boldsymbol{\varphi}_{Nk}, \Sigma_{Nk}) \cdot \Delta\boldsymbol{\varphi}_{Nk} = -\mathcal{F}_N(\boldsymbol{\varphi}_{Nk}, \Sigma_{Nk})$
2. Actualizar $\boldsymbol{\varphi}_{Nk+1} \leftarrow \boldsymbol{\varphi}_{Nk} + \Delta\boldsymbol{\varphi}_{Nk}$

Dado que la matriz de rigidez es el jacobiano de las iteraciones de Newton, la formulación discreta de la ecuación 3.68 es central tanto para problemas lineales como no lineales.

3.5.2. Medio multiplicativo con fuentes independientes

Si además de contar con fuentes independientes de fisión el medio contiene material multiplicativo, entonces los neutrones pueden provenir tanto de las fuentes independientes como de las fisiones. En este caso, tenemos que tener en cuenta la fuente de fisión, cuyo valor en la posición $\mathbf{x}$ es proporcional al flujo escalar $\phi(\mathbf{x})$. En la sección 2.1.3 indicamos que debemos utilizar expresiones diferentes para la fuente de fisión dependiendo de si estamos resolviendo un problema transitorio o estacionario. Si bien solamente una fracción $1 - \beta$ de todos los neutrones nacidos por fisión se generan en forma instantánea, en el estado estacionario debemos también sumar el resto de los β como fuente de fisión ya que suponemos el estado encontrado es un equilibrio instante a instante dado por los $1-\beta$ neutrones prompt y β neutrones retardados que provienen de fisiones operando desde un tiempo $t = -\infty$.

[25] El tamaño N' de la información relacionada con los datos de entrada Σ_N no tiene por que ser igual al tamaño N del vector solución.
[26] El término correcto es *initial guess*.

El tipo de problema discretizado es esencialmente similar al caso del medio no multiplicativo con fuentes de la sección anterior, sólo que ahora la matriz $\mathsf{A}_N(\Sigma_N)$ contiene información sobre las fuentes de fisión, que son lineales con la incógnita $\boldsymbol{\varphi}_N$. Estos casos se encuentran al estudiar sistemas sub-críticos como por ejemplo piletas de almacenamiento de combustibles gastados o procedimientos de puesta a crítico de reactores.

3.5.3. Medio multiplicativo sin fuentes independientes

En ausencia de fuentes independientes, podemos escribir las ecuaciones de transporte y difusión continuas genéricamente como [58]

$$\frac{\partial \varphi}{\partial t} = \mathcal{L}\left[\varphi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t)\right] \tag{3.69}$$

donde $\varphi = \psi$ para transporte y $\varphi = \phi$ para difusión (sin dependencia de $\hat{\mathbf{\Omega}}$), y $\mathcal{L}$ es un operador lineal homogéneo de primer orden en el espacio para transporte y de segundo orden para difusión. Esta formulación tiene infinitas soluciones de la forma

$$\varphi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \varphi_n(\mathbf{x}, \hat{\mathbf{\Omega}}, E) \cdot e^{\alpha_n \cdot t}$$

que al insertarlas en la ecuación 3.69 definen un problema de autovalores ya que

$$\begin{aligned}
\frac{\partial}{\partial t}\left[\varphi_n(\mathbf{x}, \hat{\mathbf{\Omega}}, E) \cdot e^{\alpha_n \cdot t}\right] &= \mathcal{L}\left[\varphi_n(\mathbf{x}, \hat{\mathbf{\Omega}}, E) \cdot e^{\alpha_n \cdot t}\right] \\
\alpha_n \cdot \varphi_n(\mathbf{x}, \hat{\mathbf{\Omega}}, E) \cdot e^{\alpha_n \cdot t} &= \mathcal{L}\left[\varphi_n(\mathbf{x}, \hat{\mathbf{\Omega}}, E) \cdot e^{\alpha_n \cdot t}\right] \\
\alpha_n \cdot \varphi_n(\mathbf{x}, \hat{\mathbf{\Omega}}, E) &= \mathcal{L}\left[\varphi_n(\mathbf{x}, \hat{\mathbf{\Omega}}, E)\right]
\end{aligned}$$

donde ni α_n ni φ_n dependen del tiempo t.

La solución general de la ecuación 3.69 es entonces

$$\varphi(\mathbf{x}, \hat{\mathbf{\Omega}}, E, t) = \sum_{n=0}^{\infty} C_n \cdot \varphi_n(\mathbf{x}, \hat{\mathbf{\Omega}}, E) \cdot \exp(\alpha_n \cdot t)$$

donde los coeficientes C_n son tales que satisfagan las condiciones iniciales y de contorno.

Si ordenamos los autovalores α_n de forma tal que $\mathrm{Re}(\alpha_n) \geq \mathrm{Re}(\alpha_{n+1})$ entonces para tiempos $t \gg 1$ todos los términos para $n \neq 0$ serán despreciables frente al término de φ_0. El signo de α_0 determina si la población neutrónica

a. disminuye con el tiempo ($\alpha_0 < 0$),
b. permanece constante ($\alpha_0 = 0$), o
c. aumenta con el tiempo ($\alpha_0 > 0$).

La probabilidad de que en un sistema multiplicativo sin una fuente independiente (es decir, un reactor nuclear de fisión) el primer autovalor α_0 sea exactamente cero para poder tener una solución de estado estacionario no trivial es cero. Para tener una solución matemática no nula, debemos agregar al menos un parámetro real que permita ajustar uno o más términos en forma continua para lograr ficticiamente que $\alpha_0 = 0$. Por ejemplo podríamos escribir las secciones eficaces en función de un parámetro ξ que podría ser

a. geométrico (por ejemplo la posición de una barra de control o las fugas si fuese posible modificar la geometría), o
b. físico (por ejemplo la concentración media de boro en el moderador).

De esta forma, podríamos encontrar un valor de ξ que haga que $\alpha_0 = 0$ y haya una solución de estado estacionario.

Hay un parámetro real—en el sentido matemático—que, además de permitir encontrar una solución no trivial para cualquier conjunto físicamente razonable de geometrías y secciones eficaces, nos da una idea de qué tan lejos se encuentra el modelo de la criticidad. El procedimiento consiste en dividir el término de fisiones por un número real $k_{\text{eff}} > 0$, para obtener la ecuación de difusión como

$$-\operatorname{div}\Big[D(\mathbf{x},E)\cdot\operatorname{grad}\left[\phi(\mathbf{x},E)\right]\Big]+\Sigma_t(\mathbf{x},E)\cdot\phi(\mathbf{x},E)=$$
$$\int_0^\infty \Sigma_{s_0}(\mathbf{x},E'\to E)\cdot\phi(\mathbf{x},E')\,dE'+\frac{1}{k_{\text{eff}}}\cdot\chi(E)\int_0^\infty \nu\Sigma_f(\mathbf{x},E')\cdot\phi(\mathbf{x},E')\,dE'$$

y la de transporte como

$$\hat{\mathbf{\Omega}}\cdot\operatorname{grad}\Big[\psi(\mathbf{x},\hat{\mathbf{\Omega}},E)\Big]+\Sigma_t(\mathbf{x},E)\cdot\psi(\mathbf{x},\hat{\mathbf{\Omega}},E)=$$
$$\frac{1}{4\pi}\cdot\int_0^\infty \Sigma_{s_0}(\mathbf{x},E'\to E)\cdot\int_{4\pi}\psi(\mathbf{x},\hat{\mathbf{\Omega}}',E')\,d\hat{\mathbf{\Omega}}'\,dE'+$$
$$\frac{3\cdot\hat{\mathbf{\Omega}}}{4\pi}\cdot\int_0^\infty \Sigma_{s_1}(\mathbf{x},E'\to E)\cdot\int_{4\pi}\psi(\mathbf{x},\hat{\mathbf{\Omega}}',E')\cdot\hat{\mathbf{\Omega}}'\,d\hat{\mathbf{\Omega}}'\,dE'$$
$$+\frac{1}{k_{\text{eff}}}\cdot\frac{\chi(E)}{4\pi}\int_0^\infty \nu\Sigma_f(\mathbf{x},E')\cdot\int_{4\pi}\psi(\mathbf{x},\hat{\mathbf{\Omega}}',E')\,d\hat{\mathbf{\Omega}}'\,dE'$$

La utilidad del factor k_{eff} queda reflejada en la siguiente definición.

Definición 3.27. Llamamos *factor de multiplicación efectivo* al número real k_{eff} por el cual dividimos la fuente de fisiones de las ecuaciones que modelan un medio multiplicativo sin fuentes externas. Al nuevo medio al cual se le han dividido sus fuentes de fisión por k_{eff} lo denominamos *reactor crítico asociado en k*. Si $k_{\text{eff}} > 1$ entonces el reactor original estaba super-crítico ya que hubo que disminuir sus fisiones para encontrar una solución no trivial, y viceversa. El flujo solución de las ecuaciones es el flujo del reactor crítico asociado en k y no del original, ya que si el original no estaba crítico entonces éste no tiene solución estacionaria no nula.

Al no haber fuentes independientes, todos los términos están multiplicados por la incógnita y la ecuación es homogénea. Sin embargo, ahora habrá algunos términos multiplicados por el coeficiente $1/k_{\text{eff}}$ y otros no. Una vez más, si las secciones eficaces dependen sólo de la posición $\mathbf{x}$ en forma explícita y no a través del flujo, entonces el problema es lineal y al separar en ambos miembros estos dos tipos de términos obtendremos una formulación discretizada de la forma

$$\mathsf{A}_N(\Sigma_N) \cdot \boldsymbol{\varphi}_N = \lambda_N \cdot \mathsf{B}(\Sigma_N) \cdot \boldsymbol{\varphi}_N \tag{3.70}$$

conformando un problema de autovalores generalizado, donde el autovalor λ_N dará una idea aproximada de la criticidad del reactor y el autovector $\boldsymbol{\varphi}_N$ la distribución de flujo del reactor crítico asociado en k. Si $\mathsf{B}(\Sigma_N)$ contiene los términos de fisión entonces $\lambda_N = 1/k_{\text{eff}N}$ y si $\mathsf{A}_N(\Sigma_N)$ es la que contiene los términos de fisión, entonces $\lambda = k_{\text{eff}N}$.

En general, para matrices de $N \times N$ habrá N pares autovalor-autovector. Más aún, tanto el autovalor λ_n como los elementos del autovector $\boldsymbol{\varphi}_n$ en general serán complejos. Sin embargo se puede probar [23] que, para el caso $\lambda = 1/k_{\text{eff}N}$ ($\lambda = k_{\text{eff}N}$),

1. hay un único autovalor positivo real que es mayor (menor) en magnitud que el resto de los autovalores,
2. todos los elementos del autovector correspondiente a dicho autovalor son reales y tienen el mismo signo, y
3. todos los otros autovectores o bien tienen al menos un elemento igual a cero o tienen elementos que difieren en su signo

Tanto el problema continuo como el discretizado en la ecuación 3.70 son matemáticamente homogéneos. Esta característica define dos propiedades importantes:

1. El autovector $\boldsymbol{\varphi}_N$ (es decir el flujo neutrónico) está definido a menos de una constante multiplicativa y es independiente del factor de multiplicación $k_{\text{eff}N}$. Para poder comparar soluciones debemos normalizar el flujo de alguna manera. Usualmente se define la potencia térmica total P del reactor y se normaliza el flujo de forma tal que

   $$P = \int_U \int_0^\infty e\Sigma_f(\mathbf{x}, E) \cdot \phi(\mathbf{x}, E)\, dE\, d^3\mathbf{x}$$

   donde $e\Sigma_f$ es el producto entre la la energía liberada en una fisión individual y la sección eficaz macroscópica de fisión. Si P es la potencia térmica instantánea, entonces $e\Sigma_f$ debe incluir sólo las contribuciones energéticas de los productos de fisión instantáneos. Si P es la potencia térmica total, entonces $e\Sigma_f$ debe tener en cuenta todas las contribuciones, incluyendo aquellas debidas a efectos retardados de los productos de fisión.

2. Las condiciones de contorno también deben ser homogéneas. Es decir, no es posible fijar valores de flujo o corrientes diferentes de cero.

Observación. Las secciones 5.3, 5.5, 5.6, 5.7, 5.8 y 5.10 contienen casos de este tipo de problemas.

Si, en cambio, las secciones eficaces macroscópicas dependen directa o indirectamente del flujo neutrónico (por ejemplo a través de la concentración de venenos hijos de fisión o de la temperatura de los componentes del reactor a través de la potencia disipada) entonces el problema de autovalores toma la forma no lineal

$$\mathsf{A}_N(\boldsymbol{\varphi}_N, \Sigma_N) \cdot \boldsymbol{\varphi}_N = \lambda_N \cdot \mathsf{B}(\boldsymbol{\varphi}_N, \Sigma_N) \cdot \boldsymbol{\varphi}_N$$

Existen esquemas numéricos eficientes para resolver problemas de autovalores generalizados no lineales donde la no linealidad es con respecto al autovalor λ_N [52]. Pero como en este caso la no linealidad es con el autovector $\boldsymbol{\varphi}_N$ (es decir, con el flujo) y no con el autovalor (es decir el factor de multiplicación efectivo), no son aplicables.

En el caso de neutrónica no lineal tenemos que resolver iterativamente

$$\mathsf{A}(\boldsymbol{\varphi}_{Nk}, \Sigma_{Nk}) \cdot \boldsymbol{\varphi}_{Nk+1} = \lambda_{Nk+1} \cdot \mathsf{B}(\boldsymbol{\varphi}_{Nk}, \Sigma_{Nk}) \cdot \boldsymbol{\varphi}_{Nk+1}$$

a partir de una solución inicial $\boldsymbol{\varphi}_{N0}$. En este caso el flujo está completamente determinado por la dependencia (explícita o implícita) de A y B con $\boldsymbol{\varphi}_N$ y no hay ninguna constante multiplicativa arbitraria.

Terminada la explicación del *cómo* en estos capítulos 2 y 3, pasemos entonces al *qué* en los dos capítulos que siguen.

4. Implementación computacional

> C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it, to the point where it's much much easier to generate total and utter crap with it. Quite frankly, even if the choice of C were to do *nothing* but keep the C++ programmers out, that in itself would be a huge reason to use C.
> [...]
> C++ leads to really really bad design choices. You invariably start using the "nice" library features of the language like STL and Boost and other total and utter crap, that may "help" you program, but causes:
>
> - infinite amounts of pain when they don't work [...]
> - inefficient abstracted programming models where two years down the road you notice that some abstraction wasn't very efficient, but now all your code depends on all the nice object models around it, and you cannot fix it without rewriting your app.
>
> [...]
> So I'm sorry, but for something like git, where efficiency was a primary objective, the "advantages" of C++ is just a huge mistake. The fact that we also piss off people who cannot see that is just a big additional advantage.
> *Linus Torvalds*, explaining why Git is written in C, 2007

> C++ is a badly designed and ugly language.
> It would be a shame to use it in Emacs.
> *Richard M. Stallmann*, explaining why Emacs is written in C, 2010

Hay virtualmente infinitas maneras $\aleph_0$ de diseñar un programa para que una computadora realice una determinada tarea. Y otras infinitas maneras $\aleph_1$ de implementarlas. La herramienta computacional desarrollada en esta tesis, denominada FeenoX [83] (ver Apéndice E para una explicación del nombre), fue diseñada siguiendo un patrón frecuente en la industria de software:

1. el "cliente" define un documento denominado *Sofware Requirements Specification*, y
2. el "proveedor" indica cómo cumplirá esos requerimientos en un *Sofware Design Specification*.

Una vez que ambas partes están de acuerdo, se comienza con el proyecto de ingeniería en sí con *kick-off meetings*, certificaciones de avance, órdenes de cambio, etc.

Como ya hemos mencionado, el SRS para FeenoX (apéndice A) es ficticio pero plausible. Podríamos pensarlo como un llamado a licitación por parte de una empresa, entidad pública o laboratorio nacional, para que un contratista desarrolle una herramienta computacional que permita resolver problemas matemáticos con interés práctico en aplicaciones de ingeniería. En forma muy resumida, el pliego requiere

a. buenas prácticas generales de desarrollo de software tales como

   - trazabilidad del código fuente
   - integración continua
   - posibilidad de reportar errores
   - portabilidad razonable
   - disponibilidad de dependencias adecuadas
   - documentación apropiada

b. que la herramienta pueda ser ejecutada en la nube

   - a través de ejecución remota
   - que provea mecanismos de reporte de estado
   - con posibilidad de paralelización en diferentes nodos computacionales

c. que sea aplicable a problemas de ingeniería proveyendo

   - eficiencia computacional razonable
   - flexibilidad en la definición de problemas
   - verificación y validación
   - extensibilidad

Observación. El requerimiento de paralelización está relacionado con el tamaño de los problemas a resolver y no (tanto) con la performance. Está claro que, de tener la posibilidad, resolver ecuaciones en forma paralela es más eficiente en términos del tiempo de pared[1] necesario para obtener un resultado. Pero en el SRS, la posibilidad de paralelizar el código se refiere principalmente a la capacidad de poder resolver problemas de tamaño arbitrario que no podrían ser resueltos en serie, principalmente por una limitación de la cantidad de memoria RAM.

Observación. Si bien es cierto que en teoría un algoritmo implementado en cualquier lenguaje Turing-completo podría resolver un sistema de ecuaciones algebraicas de tamaño arbitrario independientemente de la memoria RAM disponible (por ejemplo usando almacenamiento intermedio en dispositivos magnéticos o de estado sólido), no es posible obtener un resultado útil en un tiempo razonable si no se dispone de suficiente memoria RAM para evitar tener que descargar el contenido de esta memoria de alta velocidad de acceso a medios alternativos (out-of-core memory como los mencionados en el paréntesis anterior) cuya velocidad de acceso es varios órdenes de magnitud más lenta.

[1] Del inglés *wall time*.

Observación. Este esquema de definir primero los requerimientos y luego indicar cómo se los satisface evita un sesgo común dentro de las empresas de software que implica hacer algo "fácil para el desarrollador" a costa de que el usuario tenga que hacer algo "más difícil". Por ejemplo en la mayoría de los programas de elementos finitos para elasticidad lineal es necesario hacer un mapeo entre los grupos de elementos volumétricos y los materiales disponibles, tarea que tiene sentido siempre que haya más de un grupo de elementos volumétricos o más de un material disponible. Pero en los casos donde hay un único grupo de elementos volumétricos (usualmente porque se parte de un CAD con un único volumen) y un único juego de propiedades materiales (digamos un único valor E de módulo de Young y un único ν para coeficiente de Poisson), el software requiere que el usuario tenga que hacer explícitamente el mapeo aún cuando éste sea único y trivial. Un claro de ejemplo del sesgo "developer-easy/user-hard" que FeenoX no tiene[2], ya que el SRS pide que "problemas sencillos tengan inputs sencillos". En caso de que haya una única manera de asociar volúmenes geométricos a propiedades materiales, FeenoX hace la asociación implícitamente simplificando el archivo de entrada.

Según el pliego, es mandatorio que el software desarrollado sea de código abierto según la definición de la *Open Source Initiative*. El SDS (apéndice B) comienza indicando que la herramienta FeenoX es

- al software tradicional de ingeniería y
- a las bibliotecas especializadas de elementos finitos

lo que Markdown es

- a Word y
- a LaTeX

respectivamente.

Luego explica que no sólo es de código *abierto* en el sentido de la OSI sino que también es *libre* en el sentido de la *Free Software Foundation* [88], como pide el SRS. La diferencia entre código abierto y software libre es más sutil que práctica, ya que las definiciones técnicas prácticamente coinciden. El punto principal de que el código sea abierto es que permite obtener mejores resultados con mejor *performance* mientras más personas puedan estudiar el código, escrutarlo y eventualmente mejorarlo [46]. Por otro lado, el software libre persigue un fin ético relacionado con la libertad de poder ejecutar, distribuir, modificar y distribuir las modificaciones del software recibido [57], [85]. En la sección 4.4.1 discutimos los detalles de estas ideas.

Observación. Ninguno de los dos conceptos, código abierto o software libre, se refiere a la idea de *precio*. En castellano no debería haber ninguna confusión. Pero en inglés, el sustantivo adjetivado *free software* se suele tomar como gratis en lugar de libre. Si bien es cierto que la mayoría de las veces la utilización de software libre y abierto no implica el pago de ninguna tarifa directa al autor del programa, el software libre puede tener otros costos indirectos asociados.[3] El concepto importante es libertad: la *libertad* de poder contratar a uno o más programadores que modifiquen el código para que el software se comporte como uno necesita.

[2] Sabiendo que esta afirmación tiene los sesgos de confirmación, disponibilidad y supervivencia del autor.

[3] El gerente de sistemas de una empresa de ingeniería y construcción me dijo que por un tema de *costos* ellos preferían usar un servidor SQL privativo antes que uno libre: "es más caro pagarle al que sabe dónde poner el punto y coma en el archivo de configuración que pagar la licencia y configurarlo con el mouse".

Tal como como Unix[4] [50] y C[5], FeenoX es un "efecto de tercer sistema"[6] [47]. De hecho, esta diferencia entre el concepto de código abierto y software libre fue discutida en la referencia [66] durante el desarrollo de la segunda versión del sistema. Una vez más, la sección 4.4.1 provee una explicación de los conceptos fundamentales y, literalmente, de la filosofía detrás de la idea de software libre.

De las lecciones aprendidas en las dos primeras versiones, la primera un poco naíf pero funcional y la segunda más pretenciosa y compleja (apalancada en la funcionalidad de la primera), hemos convergido al diseño explicado en el SDS del apéndice B donde definimos la filosofía de diseño del software y elegimos una de estas infinitas formas de diseñar una herramienta computacional mencionadas al comienzo de este capítulo. Gran parte de este diseño está basado en la filosofía de programación Unix [47]. En el apéndice B damos ejemplos de cómo son las interfaces para definir un cierto problema y de cómo obtener los resultados. Comparamos alternativas e indicamos por qué hemos decidido diseñar el software de la forma en la que lo hemos hecho y comentamos muy superficialmente las ideas distintivas que tiene FeenoX con respecto a otros programas de elementos finitos desde el punto de vista técnico de programación. Estas características son distintivas del diseño e implementación propuestos y no son comunes. En la jerga de emprendedurismo, serían las *unfair advantages* del software con respecto a otras herramientas similares.

Observación. El código fuente de FeenoX está en Github en https://github.com/seamplex/feenox/ bajo licencia GPLv3+ (sección 4.4.1). Consiste en aproximadamente cuarenta y cinco mil líneas de código organizadas según la estructura de directorios mostrada en la figura 4.1.·

Observación. El Journal of Open Source Software ha publicado un artículo donde sucintamente se presentan las características distintivas de FeenoX con respecto a otras herramientas libres y abiertas ya existentes [83]. Tanto dicho artículo como esta tesis corresponden a la versión 1.0 del software, que a su vez corresponde al tag `v1.0` del repositorio en Github.

4.1. Arquitectura del código

Comencemos preguntándonos qué debemos tener en cuenta para implementar una herramienta computacional que permita resolver ecuaciones en derivadas parciales con aplicación en ingeniería utilizando el método de elementos finitos. Por el momento enfoquémonos en problemas lineales[7] como los analizados en los dos capítulos anteriores. Las dos tareas principales son

1. construir la matriz global de rigidez K y el vector $\mathbf{b}$ (o la matriz de masa M), y
2. resolver el sistema de ecuaciones $\mathsf{K} \cdot \mathbf{u} = \mathbf{b}$ (o $\mathsf{K} \cdot \mathbf{u} = \lambda \cdot \mathsf{M} \cdot \mathbf{u}$)
3. convertir $\mathbf{u}$ en flujos $\psi(\mathbf{x})$ y/o $\phi(\mathbf{x})$

[4] A principios de 1960, los Bell Labs en EEUU llegaron a desarrollar un sistema operativo que funcionaba bien, así que decidieron encarar MULTICS (MULTiplexed Information and Computing Service). Como terminó siendo una monstruosidad, empezaron UNIX (UNiplexed Information and Computing System) que es lo que quedó bien.

[5] A fines de 1960, también en los Bell Labs, llegaron a desarrollar un un lenguaje de programación A que funcionaba bien, así que decidieron encarar B. Como terminó siendo una monstruosidad, empezaron C que es lo que quedó bien.

[6] Del inglés *third-system effect*.

[7] Si el problema fuese no lineal o incluso transitorio, la discusión de esta sección seguiría siendo válida para la construcción de la matriz jacobiana global necesaria para aplicar cada iteración del algoritmo de Newton-Raphson (o similar).

```
.
├── dist
├── doc
├── examples
├── src
│   ├── contrib
│   ├── flow
│   ├── io
│   ├── math
│   ├── mesh
│   │   └── elements
│   ├── parser
│   └── pdes
│       ├── laplace
│       ├── mechanical
│       ├── modal
│       ├── neutron_diffusion
│       ├── neutron_sn
│       └── thermal
└── tests
```

Figura 4.1.: Estructura de directorios del código fuente de FeenoX.

Haciendo énfasis en la filosofía Unix, tenemos que escribir un programa que haga bien una sola cosa[8] que nadie más hace y que interactúe con otros que hacen bien otras cosas (regla de composición). En este sentido, nuestra herramienta se tiene que enfocar en los puntos 1 y 3. Pero tenemos que definir quién va a hacer el punto 2 para que sepamos cómo es que tenemos que construir `K` y **b**.

Las bibliotecas PETSc [7], [8] junto con la extensión SLEPc [24], [52] proveen exactamente lo que necesita una herramienta que satisfaga el SRS siguiendo la filosofía de diseño del SDS. De hecho, en 2010 seleccioné PETSc para la segunda versión del solver neutrónico por la única razón de que era una dependencia necesaria para resolver el problema de criticidad con SLEPc [72], [73]. Con el tiempo, resultó que PETSc proveía el resto de las herramientas necesarias para resolver numéricamente ecuaciones en derivadas parciales en forma portable y extensible.

Otra vez desde el punto de vista de la filosofía de programación Unix, la tarea 1 consiste en un cemento de contacto[9] entre la definición del problema a resolver por parte del ingeniero usuario y la biblioteca matemática para resolver problemas ralos[10] PETSc [42]. Cabe preguntarnos entonces cuál es el lenguaje de programación adecuado para implementar el diseño del SDS. Aún cuando ya mencionamos que cualquier lenguaje Turing-completo es capaz de resolver un sistema de ecuaciones algebraicas, está claro que no todos son igualmente convenientes. Por ejemplo Assembly o BrainFuck son interesantes en sí mismos (por diferentes razones) pero para nada útiles para la tarea que tenemos que realizar. De la misma manera, en el otro lado de la distancia con respecto al hardware, lenguajes de alto nivel como Python o R también quedan fuera de la discusión por cuestiones de efi-

[8] Do only one thing but do it well.
[9] En el sentido del inglés *glue layer*.
[10] Del inglés *sparse*.

ciencia computacional. A lo sumo, estos lenguajes interpretados podrían servir para proveer clientes finos[11] (ver sección 4.4.6) a través de APIs[12] que puedan llegar a simplificar la definición del (o los) problema(s) que tenga que resolver FeenoX. Para resumir una discusión mucho más compleja, los lenguajes candidatos para implementar la herramienta requerida por el SRS podrían ser

a. Fortran
b. C
c. C++

Según la regla de representación de Unix, la implementación debería poner la complejidad en las estructuras de datos más que en la lógica. Sin embargo, en el área de mecánica computacional, paradigmas de programación demasiado orientados a objetos impactan negativamente en la performance. La tendencia es encontrar un balance, tal como persigue la filosofía desde hace más de dos mil quinientos años a través de la virtud de la prudencia, entre programación orientada a objetos y programación orientada a datos.

Observación. En los últimos años el lenguaje Rust se ha comenzado a posicionar como una alternativa a C para *system programming*[13] debido al requerimiento intrínseco de que todos las referencias deben apuntar a una dirección de memoria virtual válida. A principios de 2023 aparecieron por primera vez líneas de código en Rust en el kernel de Linux. Pero desde el punto de vista de computación de alto rendimiento,[14] Rust (o incluso Go) no tienen nada nuevo que aportar con respecto a C.

Tal como explican los autores de PETSc (y coincidentemente Eric Raymond en [47]), C es el lenguaje que mejor se presta a este paradigma:

> Why is PETSc written in C, instead of Fortran or C++?
>
> When this decision was made, in the early 1990s, C enabled us to build data structures for storing sparse matrices, solver information, etc. in ways that Fortran simply did not allow. ANSI C was a complete standard that all modern C compilers supported. The language was identical on all machines. C++ was still evolving and compilers on different machines were not identical. Using C function pointers to provide data encapsulation and polymorphism allowed us to get many of the advantages of C++ without using such a large and more complicated language. It would have been natural and reasonable to have coded PETSc in C++; we opted to use C instead.

Fortran fue diseñado en la década de 1950 y en ese momento representó un salto cualitativo con respecto a la forma de programar las incipientes computadoras digitales [30]. Sin embargo, las suposiciones que se han tenido en cuenta con respecto al hardware y a los sistemas operativos sobre los cuales estos programas deberían ejecutarse ya no son válidas. Las revisiones posteriores como Fortran 90 son modificaciones y parches que no resuelven el problema de fondo. En cambio, C fue diseñado a principios de la década 1970 suponiendo arquitecturas de hardware y de sistemas operativos que justamente son las que se emplean hoy en día, tales como

[11] Del inglés *thin clients*.
[12] Del inglés *Application Programming Interface*.
[13] No hay traducción de este término. En el año 2008, se propuso una materia en el IB con este nombre en inglés. El consejo académico decidió traducirla y nombrarla como "programación de sistemas". Ni el nombre elegido ni el ligeramente más correcto "programación del sistema" tienen la misma denotación que el concepto original *system programming*
[14] Del inglés *high-performance computing* (HPC).

- espacio de direcciones plano[15]
- procesadores con registros de diferentes tamaños
- llamadas al sistema[16]
- entrada y salida basada en archivos
- etc.

para correr en entornos Unix, que esencialmente, es el sistema operativo de la vasta mayoría de los servidores de la nube pública, que justa y nuevamente, es lo que perseguimos en esta tesis.

Una vez más, en principio, la misma tarea puede ser implementada en cualquier lenguaje: Fortran podría implementar programación con objetos y C++ podría ser utilizado con un paradigma orientado a datos. Pero lo usual es que código escrito en Fortran sea procedural y basado en estructuras `COMMON`, resultando difícil de entender y depurar (y mantener y extender). El código escrito en C++ suele ser orientado a objetos y con varias capas de encapsulamiento, polimorfismo, métodos virtuales, re-definición de operadores[17] y contenedores enplantillados[18] resultando difícil de entender y depurar, tal como indica Linus Torvals en la cita del comienzo del capítulo. De la misma manera que el lenguaje Fortran permite realizar ciertas prácticas que bien utilizadas agregan potencia y eficiencia pero cuyo abuso lleva a código ininteligible (por ejemplo el uso y abuso de bloques `COMMON`), C++ permite ciertas prácticas que bien utilizadas agregan potencia pero cuyo abuso lleva a código de baja calidad (por ejemplo el uso y abuso de `templates`, `shared_pointers`, `binds`, `moves`, lambdas, wrappers sobre wrappers, objetos sobre objetos, interfaz sobre interfaz, etc.), a veces en términos de eficiencia, a veces en términos del concepto de Unix *compactness* [47]:

> Compactness is the property that a design can fit inside a human being's head. A good practical test for compactness is this: Does an experienced user normally need a manual? If not, then the design (or at least the subset of it that covers normal use) is compact.

y usualmente en términos de la regla de simplicidad en la filosofía de Unix:

> Add complexity only where you must.

En efecto, Fortran y C++ hacen fácil agregar complejidad innecesaria. En C, no es fácil agregar complejidad innecesaria. Por lo tanto, como bien dice Linus Torvalds una vez más en la cita del comienzo del capítulo, si con esta decisión lo único que pudiésemos hacer es evitar que se pueda agregar gratuitamente complejidad al código, esa ya sería una razón suficiente para tomarla.

4.1.1. Construcción de los objetos globales

Habiendo decidido entonces construir la matriz `K` y el vector **b** como una "glue layer" implementada en C utilizando una estructura de datos que PETSc pueda entender, preguntémonos ahora qué necesitamos para construir estos objetos. Para simplificar el argumento, supongamos por ahora que queremos resolver la ecuación generalizada de Poisson de la sección 3.4.1. La matriz global `K` proviene

[15] Del inglés *flat address space*.
[16] Del inglés *system calls*.
[17] Del inglés *operator overloading*.
[18] Del inglés *templated containers*.

de ensamblar las matrices elementales K_i para todos los elementos volumétricos e_i según la definición 3.23. De la misma manera, el vector global $\mathbf{b}_i$ proviene de ensamblar las contribuciones elementales $\mathbf{b}_i$ tanto de los elementos volumétricos (definición 3.24) como de los elementos de superficie con condiciones de contorno naturales (definición 3.25).

foreach *elemento volumétrico* e_i **do**
  $\mathsf{K}_i \leftarrow 0$
  $\mathbf{b}_i \leftarrow 0$
  foreach *punto de cuadratura* $q = 1, \dots, Q_i$ **do**
    $\mathsf{J}_i(\boldsymbol{\xi}_q) \leftarrow \mathsf{B}_c(\boldsymbol{\xi}_q) \cdot \mathsf{C}_i$ /* ecuación 3.51 */
    $\mathsf{B}_i(\boldsymbol{\xi}_q) \leftarrow \mathsf{J}_i^{-T}(\boldsymbol{\xi}_q) \cdot \mathsf{B}_c(\boldsymbol{\xi}_q)$ /* ecuación 3.52 */
    $\mathbf{x}_q(\boldsymbol{\xi}_q) \leftarrow \sum_{j=1}^{J_i} h_j(\boldsymbol{\xi}_q) \cdot \mathbf{x}_j$ /* ecuación 3.50 */
    $\mathsf{K}_i \leftarrow \mathsf{K}_i + \omega_q \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right| \left\{\mathsf{B}_i^T(\boldsymbol{\xi}_q) \cdot k(\mathbf{x}_q) \cdot \mathsf{B}_i(\boldsymbol{\xi}_q)\right\}$ /* ecuación 3.49 */
    $\mathbf{b}_i \leftarrow \mathbf{b}_i + \omega_q \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right| \left\{\mathsf{H}_c^T(\boldsymbol{\xi}_q) \cdot f(\mathbf{x}_q)\right\}$ /* ecuación 3.53 */
  end
  ensamblar $\mathsf{K}_i \rightarrow \mathsf{K}$
  ensamblar $\mathbf{b}_i \rightarrow \mathbf{b}$
end
foreach *elemento superficial* e_i *con condición de Neumann* $p(\mathbf{x})$ **do**
  $\mathbf{b}_i \leftarrow 0$
  foreach *punto de cuadratura* $q = 1, \dots, Q_i$ **do**
    $\mathbf{b}_i \leftarrow \mathbf{b}_i + \omega_q^{(D-1)} \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right| \cdot \left\{\mathsf{H}_{c'}^T(\boldsymbol{\xi}_q) \cdot p(\mathbf{x}_q)\right\}$ /* ecuación 3.54 */
  end
  ensamblar $\mathbf{b}_i \rightarrow \mathbf{b}$
end

Algoritmo 2: Posible implementación de alto nivel de la construcción de K y $\mathbf{b}$ para el problema de Poisson generalizado de la sección 3.4.1

Una posible implementación en pseudo código de alto nivel de esta construcción podría ser la ilustrada en el algoritmo 2. La aplicación de las condiciones de contorno de Dirichlet según la discusión de la sección 3.4.1.2 puede ser realizada con el algoritmo 3 o con el algoritmo 4. En un caso barremos sobre elementos y tenemos que encontrar el índice global asociado a cada nodo local. En otro caso barremos sobre nodos globales pero tenemos que encontrar los elementos asociados al nodo global.

La primera conclusión que podemos extraer es que para problemas escalares, la ecuación a resolver está determinada por la expresión entre llaves de los términos evaluados en cada punto de Gauss para K_i y para $\mathbf{b}_i$. Si el problema tuviese más de una incógnita por nodo y reemplazamos las matrices H_c y B_i por sus versiones G-aware H_{Gc} y B_{Gi}, entonces otra vez la ecuación a resolver está completamente definida por la expresión entre llaves.

Por ejemplo, si quisiéramos resolver difusión de neutrones multigrupo tendríamos que reemplazar el término entre llaves $\mathsf{B}_i^T(\boldsymbol{\xi}_q) \cdot k(\mathbf{x}_q) \cdot \mathsf{B}_i(\boldsymbol{\xi}_q)$ en la ecuación 3.49 por $\mathsf{L}_i + \mathsf{A}_i - \mathsf{F}_i$ para obtener la ecuación 3.62

foreach *elemento superficial* e_i *con condición de Dirichlet* $g(\mathbf{x})$ **do**
  for *cada nodo local* $j = 1, \dots, J_i$ **do**
    calcular la fila global k correspondiente al nodo local j del elemento e_i
    hacer cero la fila k de la matriz global K
    poner un uno en la diagonal de la fila k de la matriz global K
    poner $g(\mathbf{x}_j)$ en la fila k del vector global $\mathbf{b}$
  end
end

Algoritmo 3: Una forma de poner condiciones de Dirichlet en el problema discretizado, barriendo sobre elementos y calculando la fila global a partir del nodo local.

foreach *nodo global* $j = 1, \dots J$ **do**
  foreach *elemento* e_i *al que pertenece el nodo global* j **do**
    if *el elemento* e_i *tiene una condición de Dirichlet* **then**
      hacer cero la fila j de la matriz global K
      poner un uno en la diagonal de la fila j de la matriz global K
      poner $g(\mathbf{x}_j)$ en la fila j del vector global $\mathbf{b}$
    end
  end
end

Algoritmo 4: Otra forma de poner condiciones de Dirichlet en el problema discretizado, barriendo sobre nodos globales y encontrando los elementos asociados.

$$\mathsf{K}_i \leftarrow \mathsf{K}_i + \omega_q \cdot \left|\det\left[\mathsf{J}_i\left(\boldsymbol{\xi}_q\right)\right]\right| \cdot \left\{\mathsf{L}_i(\boldsymbol{\xi}_q) + \mathsf{A}_i(\boldsymbol{\xi}_q) - \mathsf{F}_i(\boldsymbol{\xi}_q)\right\}$$

con las matrices intermedias según la ecuación 3.63

$$\begin{aligned} \mathsf{L}_i &= \mathsf{B}_{Gi}^T(\boldsymbol{\xi}_q) \cdot \mathsf{D}_D(\boldsymbol{\xi}_q) \cdot \mathsf{B}_{Gi}(\boldsymbol{\xi}_q) \\ \mathsf{A}_i &= \mathsf{H}_{Gc}^T(\boldsymbol{\xi}_q) \cdot \mathsf{R}(\boldsymbol{\xi}_q) \cdot \mathsf{H}_{Gc}(\boldsymbol{\xi}_q) \\ \mathsf{F}_i &= \mathsf{H}_{Gc}^T(\boldsymbol{\xi}_q) \cdot \mathsf{X}(\boldsymbol{\xi}_q) \cdot \mathsf{H}_{Gc}(\boldsymbol{\xi}_q) \end{aligned} \tag{3.63}$$

más las contribuciones a la matriz de rigidez para condiciones de Robin. Pero en principio podríamos escribir un algoritmo genérico (¿una cáscara?) que implemente el método de elementos finitos para resolver una cierta ecuación diferencial completamente definida por la expresión dentro de las llaves (¿pasadas como argumentos?).

La segunda conclusión proviene de preguntarnos qué es lo que necesitan los algoritmos 2, 3 y 4 para construir la matriz de rigidez global K y el vector $\mathbf{b}$:

i. los conjuntos de cuadraturas de gauss $\omega_q, \boldsymbol{\xi}_q$ para el elemento e_i
ii. las matrices canónicas H_c, B_c y H_{Gc}
iii. las matrices elementales C_i, B_{Gi},
iv. poder evaluar en $\mathbf{x}_q$

    a. las conductividad $k(\mathbf{x})$ (o las secciones eficaces para construir $\mathsf{D}_D(\boldsymbol{\xi}_q)$, $\mathsf{R}(\boldsymbol{\xi}_q)$ y $\mathsf{X}(\boldsymbol{\xi}_q)$)
    b. la fuente volumétrica $f(\mathbf{x})$ (o las fuentes independientes isotrópicas $s_{0,g}(\boldsymbol{\xi}_q)$)

c. las condiciones de contorno $p(\mathbf{x})$ y $g(\mathbf{x})$ (o las correspondientes a difusión de neutrones)

Los tres primeros puntos no dependen de la ecuación a resolver. Lo único que se necesita para evaluarlos es tener disponible la topología de la malla no estructurada que discretiza el dominio $U \in \mathbb{R}^D$ a través la posición $\mathbf{x}_j \in \mathbb{R}^D$ de los nodos y la conectividad de cada uno de los elementos e_i.

El cuarto en principio sí depende de la ecuación, pero finalmente se reduce a la capacidad de evaluar

1. propiedades de materiales
2. condiciones de contorno

en una ubicación espacial arbitraria $\mathbf{x}$.

Para fijar ideas, supongamos que tenemos un problema de conducción de calor. La conductividad $k(\mathbf{x})$[19] o en general, cualquier propiedad material puede depender de la posición $\mathbf{x}$ porque

a. existen materiales con propiedades discontinuas en diferentes ubicaciones $\mathbf{x}$, y/o
b. la propiedad depende continuamente de la posición aún para el mismo material.

De la misma manera, una condición de contorno (sea ésta esencial o natural) puede depender discontinuamente con la superficie donde esté aplicada o continuamente dentro de la misma superficie a través de alguna dependencia con la posición $\mathbf{x}$.

Entonces, la segunda conclusión es que si nuestra herramienta fuese capaz de proveer un mecanismo para definir propiedades materiales y condiciones de contorno que puedan depender

a. discontinuamente según el volumen o superficie al que pertenezca cada elemento (algunos elementos volumétricos pertenecerán al combustible y otros al moderador, algunos elementos superficiales pertenecerán a una condición de simetría y otros a una condición de vacío), y/o
b. continuamente en el espacio según variaciones locales (por ejemplo cambios de temperatura y/o densidad, concentración de venenos, etc.)

entonces podríamos resolver ecuaciones diferenciales arbitrarias discretizadas espacialmente con el método de elementos finitos.

4.1.2. Polimorfismo con apuntadores a función

Según la discusión de la sección anterior, podemos diferenciar entre dos partes del código:

1. Una que tendrá que realizar tareas "comunes"
    - en el sentido de que son las mismas para todas las PDEs tal como leer la malla y evaluar funciones en un punto $\mathbf{x}$ arbitrario del espacio
2. Otra parte que tendrá que realizar tareas particulares para cada ecuación a resolver

[19] Si la conductividad dependiera de la temperatura T el problema sería no lineal. Pero en el paso k e la iteración de Newton tendríamos $k(\mathbf{T}_k(\mathbf{x}) = k_k(\mathbf{x})$ por lo que la discusión sigue siendo válida.

- por ejemplo evaluar las expresiones entre llaves en el q-ésimo punto de Gauss del elemento i-ésimo

Definición 4.1 (framework). Llamamos *framework* a la parte del código que implementa las tareas comunes.

Por decisión de diseño, el problema que FeenoX tiene que resolver tiene que estar completamente definido en el archivo de entrada. Esto es, no se debe necesitar intervención del usuario luego de la ejecución. Entonces, este archivo de entrada debe justamente definir qué clase de ecuación se debe resolver. Como el tipo de ecuación se lee en tiempo de ejecución, el framework debe poder ser capaz de llamar a una u otra (u otra) función que le provea la información particular que necesita: por ejemplo las expresiones entre llaves para la matriz de rigidez y para las condiciones de contorno.

Una posible implementación (ingenua) sería hacer, para cada punto de Gauss de cada uno de los elementos,

if *la PDE es la ecuación de Poisson* **then**
  evaluar $\mathtt{B}_i^T \cdot k(\mathbf{x}_q) \cdot \mathtt{B}_i$ `/* ecuación 3.49 */`
else if *la PDE es difusión de neutrones* **then**
  evaluar $\mathtt{L}_i(\boldsymbol{\xi}_q) + \mathtt{A}_i(\boldsymbol{\xi}_q) + \mathtt{F}_i(\boldsymbol{\xi}_q)$ `/* ecuación 3.62 */`
else if *la PDE es S_N* **then**
  evaluar $\mathtt{L}_i(\boldsymbol{\xi}_q) + \mathtt{A}_i(\boldsymbol{\xi}_q) + \mathtt{F}_i(\boldsymbol{\xi}_q)$ `/* ecuación 3.65 */`
else
  quejarse "no sé resolver esta PDE"
end

De la misma manera, necesitaríamos bloques `if` (o `switch`) de este tipo para

- inicializar el problema
- evaluar condiciones de contorno
- calcular resultados derivados (por ejemplo flujos de calor a partir de las temperaturas, tensiones a partir de desplazamientos o corrientes a partir de flujos neutrónicos)
- etc.

Está claro que esto es

1. feo,
2. ineficiente, y
3. difícil de extender

En C++, podríamos diseñar una mejor implementación mediante una jerarquía de clases donde las clases hijas implementarían métodos virtuales que el framework llamaría cada vez que necesite evaluar el término entre llaves. Si bien C no tiene "métodos virtuales", sí tiene apuntadores a función (que es justamente lo que PETSc usa para implementar polimorfismo como mencionamos en la página 166) por lo que podemos usar este mecanismo para lograr una implementación superior, que explicamos a continuación.

Por un lado, sí existe un lugar del código con un bloque `if` según el tipo de PDE requerida en tiempo de ejecución que consideramos feo, ineficiente y difícil de extender. Pero,

a. este *único* bloque de condiciones `if` se ejecutan una sola vez en el momento de analizar gramaticalmente[20] el archivo de entrada y lo que hacen es resolver un apuntador a función a la dirección de memoria de una rutina de inicialización particular que el framework debe llamar antes de comenzar a construir `K` y `b`:

```
if (strcasecmp(token, "laplace") == 0) {
  feenox.pde.init_parser_particular = feenox_problem_init_parser_laplace;
} else if (strcasecmp(token, "mechanical") == 0) {
  feenox.pde.init_parser_particular = feenox_problem_init_parser_mechanical;
} else if (strcasecmp(token, "modal") == 0) {
  feenox.pde.init_parser_particular = feenox_problem_init_parser_modal;
} else if (strcasecmp(token, "neutron_diffusion") == 0) {
  feenox.pde.init_parser_particular = feenox_problem_init_parser_neutron_diffusion;
} else if (strcasecmp(token, "neutron_sn") == 0) {
  feenox.pde.init_parser_particular = feenox_problem_init_parser_neutron_sn;
} else if (strcasecmp(token, "thermal") == 0) {
  feenox.pde.init_parser_particular = feenox_problem_init_parser_thermal;
} else {
  feenox_push_error_message("unknown problem type '%s'", token);
  return FEENOX_ERROR;
}
```

Estas funciones de inicialización a su vez resuelven los apuntadores a función particulares para evaluar contribuciones elementales volumétricas en puntos de Gauss, condiciones de contorno, post-procesamiento, etc.

b. El bloque `if` mostrado en el punto anterior es generado programáticamente a partir de un script (regla de Unix de generación, sección C.14) que analiza (*parsea*) el árbol del código fuente y, para cada subdirectorio en `src/pdes`, genera un bloque `if` automáticamente. Es fácil ver el patrón que siguen cada una de las líneas del listado en el punto a y escribir un script o macro para generarlo programáticamente. De hecho en la sección 4.1.4 mostramos una implementación simplificada de dicho script.

Entonces,

1. Si bien ese bloque sigue siendo feo, es generado y compilado por una máquina que no tiene el mismo sentido estético que nosotros.

2. Reemplazamos la evaluación de n condiciones `if` para llamar a una dirección de memoria fija para cada punto de Gauss para cada elemento por una des-referencia de un apuntador a función. En términos de eficiencia, esto es similar (tal vez más eficiente) que un método virtual de C++. Esta des-referencia dinámica no permite que el compilador pueda hacer un `inline` de la función llamada, pero el gasto extra[21] es muy pequeño. En cualquier caso, el mismo script que "parsea" la estructura en `src/pdes` podría modificarse para generar un binario de FeenoX para cada PDE donde en lugar de llamar a un apuntador a función se llame directamente a las funciones propiamente dichas permitiendo optimización en tiempo de vinculación[22] que le permita al compilador hacer el `inline` de la función particular (ver sección 4.4.4).

[20] Del inglés *parse*.
[21] Del inglés *overhead*.
[22] Del inglés *link-time optimization*.

3. El script que parsea la estructura de `src/pdes` en busca de los tipos de PDEs disponibles es parte del paso `autogen.sh` (ver la discusión de la sección 4.1.4) dentro del esquema `configure` + `make` de Autotools. Las PDEs soportadas por FeenoX puede ser extendidas agregando un nuevo subdirectorio dentro de `src/pdes` donde ya existen

   - `laplace`
   - `thermal`
   - `mechanical`
   - `modal`
   - `neutron_diffusion`
   - `neutron_sn`

   tomando uno de estos subdirectorios como plantilla.[23] De hecho también es posible eliminar completamente uno de estos directorios en el caso de no querer que FeenoX pueda resolver alguna PDE en particular. De esta forma, `autogen.sh` permite extender (o reducir) la funcionalidad del código, que es uno de los puntos solicitados en el SDS. Más aún, sería posible utilizar este mecanismo para cargar funciones particulares desde objetos compartidos[24] en tiempo de ejecución (utilizando la función `dlopen` por ejemplo), incrementando aún más la extensibilidad de la herramienta.

4.1.3. Definiciones e instrucciones

En la sección 1.2 mencionamos (y en el apéndice B explicamos en detalle) que la herramienta desarrollada es una especie de "función de transferencia" entre uno o más archivos de entrada y cero o más archivos de salida (incluyendo la salida estándar `stdout`). En la figura 4.2 ilustramos nuevamente esta idea para un cierto problema particular, digamos el Benchmark 3D PWR de IAEA mencionado en el capítulo 1. Este archivo de entrada, que a su vez puede incluir otros archivos de entrada y/o hacer referencia a otros archivos de datos (incluyendo la malla en formato `.msh`) contiene palabras clave[25] en inglés que, por decisión de diseño, deben

1. definir completamente el problema de resolver
2. ser lo más auto-descriptivas posible
3. permitir una expresión algebraica en cualquier lugar donde se necesite un valor numérico
4. mantener una correspondencia uno a uno entre la definición "humana" del problema y el archivo de entrada
5. hacer que el archivo de entrada de un problema simple sea simple

Estas keywords pueden ser

a. definiciones[26] (sustantivos)

   - qué PDE hay que resolver

[23] Del inglés *template*
[24] Del inglés *shared objects*
[25] Del inglés *keywords*.
[26] La etimología de la palabra "definición" proviene de "dar fin", es decir, "dar límite" o delimitar qué es lo que queda dentro de la definición y qué no.

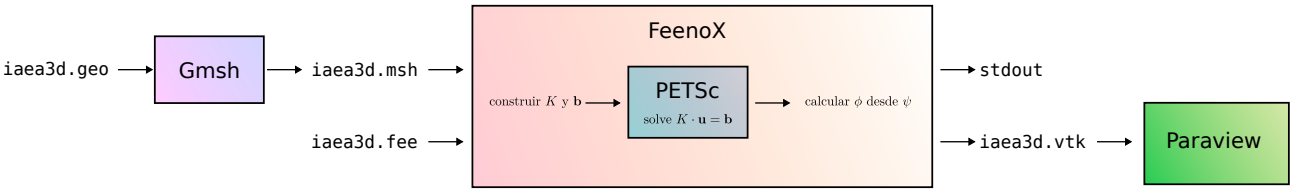


Figura 4.2.: Flujo de información a través de FeenoX para el problema 3D PWR Benchmark de la figura 1.2.

- propiedades materiales
- condiciones de contorno
- variables, vectores y/o funciones auxiliares
- etc.

b. instrucciones (verbos)

- leer la malla
- resolver problema
- asignar valores a variables
- calcular integrales o encontrar extremos sobre la malla
- escribir resultados
- etc.

Por un lado, las definiciones se realizan a tiempo de parseo. Por otro lado, Una vez que FeenoX acepta que el archivo de entrada es válido, comienza la ejecución de las instrucciones en el orden indicado en el archivo de entrada. De hecho, FeenoX tiene un apuntador a instrucción[27] que recorre una lista a medida que avanza la ejecución de las instrucciones, pero teniendo en cuenta que existe palabras clave (como `IF` y `WHILE`) que permiten flujos de ejecución no triviales, especialmente en problemas iterativos, cuasi-estáticos o transitorios:

```
// sweep the first & last range but minding the conditional blocks
instruction_t *ip = first;
while (ip != last) {
  feenox_call(ip->routine(ip->argument));

  if (feenox.next_instruction != NULL) {
    ip = feenox.next_instruction;
    feenox.next_instruction = NULL;
  } else {
    ip = ip->next;
  }
}
```

Por ejemplo, en

```
f(x) = x^2
PRINT f(1/2) f(1) f(2)
```

la primera línea es una definición (dada por el signo `=`) y la segunda es una instrucción (verbo `PRINT`).

En particular, la lectura del archivo de malla es una instrucción y no una definición porque

[27] Del inglés *instruction pointer*.

i. el nombre del archivo de la malla puede depender de alguna variable cuyo valor deba ser evaluado en tiempo de ejecución con una instrucción previa. Por ejemplo

```
INPUT_FILE surprise PATH nafems-le1%g.msh round(random(0,1))
READ_MESH surprise
PRINT cells
```

```
$ feenox surprise.fee
18700
$ feenox surprise.fee
18700
$ feenox surprise.fee
32492
$ feenox surprise.fee
32492
$ feenox surprise.fee
18700
$
```

ii. el archivo con la malla en sí puede ser creado internamente por FeenoX con una instrucción previa `WRITE_MESH` y/o modificada en tiempo de ejecución

```
READ_MESH square.msh SCALE 1e-3
WRITE_MESH square_tmp.vtk

INPUT_FILE tmp PATH square_tmp.vtk
READ_MESH tmp
```

iii. en problemas complejos puede ser necesario leer varias mallas antes de resolver la PDE en cuestión, por ejemplo leer una distribución de temperaturas en una malla gruesa de primer orden para utilizarla al evaluar las propiedades de los materiales de la PDE que se quiere resolver. Ver sección 5.1 y sección B.3.2.2.

Comencemos con un problema sencillo para luego agregar complejidad en forma incremental. A la luz de la discusión de este capítulo, preguntémonos ahora qué necesitamos para resolver un problema de conducción de calor estacionario sobre un dominio $U \in \mathbb{R}^D$ con un único material:

1. definir que la PDE es conducción de calor en D dimensiones
2. leer la malla con el dominio $U \in \mathbb{R}^D$ discretizado
3. definir la conductividad térmica $k(\mathbf{x})$ del material
4. definir las condiciones de contorno del problema
5. construir K y $\mathbf{b}$
6. resolver $\mathsf{K} \cdot \mathbf{u} = \mathbf{b}$
7. hacer lo que nos haya pedido el usuario en el archivo de entrada con los resultados, por ejemplo
   - calcular T_{max}
   - calcular T_{mean}
   - calcular el vector flujo de calor $q''(\mathbf{x}) = -k(\mathbf{x}) \cdot \nabla T$
   - comparar con soluciones analíticas
   - etc.
8. escribir los resultados

El problema de conducción de calor más sencillo es un slab unidimensional en el intervalo $x \in [0, 1]$ con conductividad uniforme y condiciones de Dirichlet $T(0) = 0$ y $T(1) = 1$ en ambos extremos. Por diseño, el archivo de entrada tiene que ser sencillo (porque dijimos que el problema es sencillo, en el capítulo 5 vamos a ver problemas más complicados con inputs complicados) y tener una correspondencia uno a uno con la definición "humana" del problema:

```
PROBLEM thermal 1D              # 1. definir que la PDE es calor 1D
READ_MESH slab.msh              # 2. leer la malla
k = 1                           # 3. definir conductividad uniforme igual a uno
BC left  T=0                    # 4. condiciones de contorno de Dirichlet
BC right T=1                    #    "left" y "right" son nombres en la malla
SOLVE_PROBLEM                   # 5. y 6. construir y resolver
PRINT T(0.5)                    # 7. y 8. escribir en stdout la temperatura en x=0.5
```

En este caso sencillo, la conductividad k fue dada como una variable `k` con un valor igual a uno. Estrictamente hablando, la asignación fue una instrucción. Pero en el momento de resolver el problema, las funciones particulares de la PDE de conducción de calor (dentro de `src/pdes/thermal`) buscan una variable llamada `k` y toman su valor para utilizarlo idénticamente en todos los puntos de Gauss de los elementos volumétricos al calcular el término $\mathtt{B}_i^T \cdot k \cdot \mathtt{B}_i$.

Si la conductividad no fuese uniforme sino que dependiera del espacio por ejemplo como

$$k(x) = 1 + x$$

entonces el archivo de entrada sería

```
PROBLEM thermal 1D
READ_MESH slab.msh
k(x) = 1+x                      # 3. conductividad dada por una función de x
BC left  T=0
BC right T=1
SOLVE_PROBLEM
PRINT T(1/2) log(1+1/2)/log(2)  # 7. y 8. imprimir el valor numérico y la solución analítica
```

En este caso, no hay una variable llamada `k` sino que hay una *función* de x (definida con una definición) con la expresión algebraica $1 + x$. Entonces la rutina particular dentro de `src/pde/thermal` que evalúa las contribuciones volumétricas elementales de la matriz de rigidez toma dicha función $k(x)$ como la propiedad "conductividad" y la evalúa como $\mathtt{B}_i^T(\mathbf{x}_q) \cdot k(\mathbf{x}_q) \cdot \mathtt{B}_i(\mathbf{x}_q)$. La salida, que por diseño está 100% definida por el archivo de entrada (reglas de Unix de silencio sección C.11 y de economía sección C.13) consiste en la temperatura evaluada en $x = 1/2$ junto con la solución analítica $\log(1 + \frac{1}{2})/\log(2)$ en dicho punto.

Por completitud, mostramos que también la conductividad podría depender de la temperatura. En este caso particular el problema resulta no lineal. Mencionamos algunas particularidades sin ahondar en detalles. El parser algebraico de FeenoX sabe que k depende de T, por lo que la rutina particular de inicialización de la PDE de conducción de calor marca que el problema debe ser resuelto por PETSc con un objeto SNES en lugar de un KSP como para el caso lineal. FeenoX también calcula el jacobiano necesario para resolver el problema con un método de Newton iterativo, que incluye un término proporcional a $\partial k/\partial T$:

```
PROBLEM thermal 1D
READ_MESH slab.msh
k(x) = 1+T(x)                    # 3. la conductividad ahora depende de T(x)
BC left  T=0
BC right T=1
SOLVE_PROBLEM
PRINT T(1/2) sqrt(1+(3*0.5))-1
```

La ejecución de FeenoX sigue también las reglas tradicionales de Unix. Se debe proveer la ruta al archivo de entrada como principal argumento luego del ejecutable. Es un argumento y no como una opción ya que la funcionalidad del programa depende de que se indique un archivo de entrada, por lo que no es "opcional". Sí se pueden agregar opciones siguiendo las reglas POSIX. Algunas opciones son para FeenoX (por ejemplo `--progress` o `--elements_info` y otras son pasadas a PETSc/SLEPc (por ejemplo `--ksp_view` o `--mg_levels_pc_type=sor`). Al ejecutar los tres casos anteriores, obtenemos los resultados solicitados con la instrucción `PRINT` en la salida estándar:

```
$ feenox thermal-1d-dirichlet-uniform-k.fee
0.5
$ feenox thermal-1d-dirichlet-space-k.fee
0.584945        0.584963
$ feenox thermal-1d-dirichlet-temperature-k.fee
0.581139        0.581139
$
```

Para el caso de conducción de calor estacionario solamente hay una única propiedad cuyo nombre debe ser `k` para que las rutinas particulares la detecten como la conductividad k que debe aparecer en la contribución volumétrica. Si el problema tuviese dos materiales diferentes, digamos `A` y `B` (es decir, en la malla hubiera algunos elementos asociados a la etiqueta volumétrica `A` y otros a la etiqueta `B`) entonces habría dos maneras de definir sus propiedades materiales en FeenoX:

1. agregando el sufijo `_A` y `_B` a la variable `k` o a la función `k(x)`, es decir

   ```
   k_A = 1
   k_B = 2
   ```

   si $k_A = 1$ y $k_B = 2$, o

   ```
   k_A(x) = 1+2*x
   k_B(x) = 3+4*x
   ```

   si $k_A(x) = 1 + 2x$ y $k_B(x) = 3 + 4x$.

2. utilizando una palabra clave `MATERIAL` (definición) para ´cada material, del siguiente modo

   ```
   MATERIAL A k=1
   MATERIAL B k=2
   ```

   o

   ```
   MATERIAL A k=1+2*x
   MATERIAL B k=3+4*x
   ```

De esta forma, el framework implementa las dos posibles dependencias de las propiedades de los materiales discutidasen la página 170:

a. continua con el espacio a través de expresiones algebraicas que pueden involucrar funciones definidas por puntos en interpoladas como discutimos en la sección 4.3, y
b. discontinua según el material al que pertenece el elemento.

Con respecto a las condiciones de contorno, la lógica es similar pero ligeramente más complicada. Mientras que a partir del nombre de la propiedad material (incluyendo las fuentes volumétricas) las rutinas particulares pueden evaluar las contribuciones volumétricas, sean a la matriz de rigidez `K` a través de la conductividad `k` o al vector **b** a través de la fuente de calor por unidad de volumen `q`, para las condiciones de contorno se necesita un poco más de información. Supongamos que una superficie tiene una condición de "simetría" y que queremos que la línea del archivo de entrada que la defina sea

```
BC left  symmetry
```

donde `left` es el nombre de la entidad superficial definida en la malla y `symmetry` es una palabra clave que indica qué tipo de condición de contorno tienen los elementos que pertenecen a `left`. La forma de implementar numéricamente (e incluso físicamente) esta condición de contorno depende de la PDE que estamos resolviendo. No es lo mismo poner una condición de simetría en las ecuaciones de

- poisson generalizada
- elasticidad lineal
- difusión de neutrones
- transporte por S_N

Entonces, si bien las propiedades de materiales pueden ser parseadas por el framework y aplicadas por las rutinas particulares, las condiciones de contorno deben ser parseadas y aplicadas por las rutinas particulares. De todas maneras, la lógica es similar: las rutinas particulares proveen puntos de entrada[28] que el framework llama en los momentos pertinentes durante la ejecución de las instrucciones.

Para terminar de ilustrar la idea, consideremos el problema de Reed (discutido en detalle en la sección 5.2) que propone resolver con S_N un slab uni-dimensional compuesto por varios materiales, algunos con una fuente independiente, otros sin fuente e incluso un material de vacío:

```
#
#     |         |    |         |    |         |
#  m  | src= 50 | 0  |    0    | 1  |    0    |    v
#  i  |         |    |         |    |         |    a
#  r  | tot= 50 | 5  |    0    | 1  |    1    |    c
#  r  |         |    |         |    |         |    u
#  o  | scat=0  | 0  |    0    | 0.9|   0.9   |    u
#  r  |         |    |         |    |         |    m
#     |         |    |         |    |         |
#     |    1    | 2  |    3    | 4  |    5    |
#     |         |    |         |    |         |
#     +---------+----+---------+----+---------+-------> x
```

[28] Del inglés *entry points*.

```
#     x=0        x=2   x=3        x=5   x=6        x=8

PROBLEM neutron_sn DIM 1 GROUPS 1 SN $1

READ_MESH reed.msh

MATERIAL source_abs    S1=50 Sigma_t1=50 Sigma_s1.1=0
MATERIAL absorber      S1=0  Sigma_t1=5  Sigma_s1.1=0
MATERIAL void          S1=0  Sigma_t1=0  Sigma_s1.1=0
MATERIAL source_scat   S1=1  Sigma_t1=1  Sigma_s1.1=0.9
MATERIAL reflector     S1=0  Sigma_t1=1  Sigma_s1.1=0.9

BC left  mirror
BC right vacuum

SOLVE_PROBLEM

FUNCTION ref(x) FILE reed-ref.csv INTERPOLATION steffen
PRINT sqrt(integral((ref(x)-phi1(x))^2,x,0,8))/8
```

- La primera línea es una definición (`PROBLEM` es un sustantivo) que le indica a FeenoX que debe resolver las ecuaciones S_N en $D = 1$ dimensión con $G = 1$ grupo de energías y utilizando una discretización angular N dada por el primer argumento $1 de la línea de comando luego del nombre del archivo de entrada. De esta manera se pueden probar diferentes discretizaciones con el mismo archivo de entrada, digamos `reed.fee`

  ```
  $ feenox reed 2   # <- S2
  [...]
  $ feenox reed 4   # <- S4
  [...]
  ```

- La segunda línea es una instrucción (el verbo `READ`) que indica que FeenoX debe leer la malla donde resolver problema del archivo `reed.msh`. Si el archivo de entrada se llamara `reed.fee`, esta línea podría haber sido `READ_MESH $0.msh`. En caso de leer varias mallas, la que define el dominio de la PDE es

  a. la primera de las instrucciones `READ_MESH`, o
  b. la definida explícitamente con la palabra clave `MAIN_MESH`

- El bloque de palabras clave `MATERIAL` definen (con un sustantivo) las secciones eficaces macroscópicas (`Sigma`) y las fuentes independientes (`S`). En este caso todas las propiedades son uniformes dentro de cada material.

- Las siguientes dos líneas definen las condiciones de contorno (`BC` es la abreviación *boundary condition* que es un sustantivo adjetivado): la superficie `left` tiene una condición de simetría (o espejo) y la superficie `right` tiene una condición de vacío.

- La instrucción `SOLVE_PROBLEM` le pide a FeenoX que

  1. construya la matriz global `K` y el vector **b** (pidiéndole a su vez el integrando a las rutinas específicas del subdirectorio `src/pdes/neutron_sn`)

2. que le pida a su vez a PETSc que resuelva el problema $\mathtt{K} \cdot \mathbf{u} = \mathbf{b}$ (como hay fuentes independientes entonces la rutina de inicialización del problema `neutron_sn` sabe que debe resolver un problema lineal, si no hubiese fuentes y sí secciones eficaces de fisión se resolvería un problema de autovalores con la biblioteca SLEPc)
3. que extraiga los valores nodales de la solución $\mathbf{u}$ y defina las funciones del espacio $\psi_{mg}(x)$ y $\phi_g(x)$. Si `$1` fuese `2` entonces las funciones con la solución serían
   - `psi1.1(x)`
   - `psi1.2(x)`
   - `phi1(x)`

   Si `$1` fuese `4` entonces serían
   - `psi1.1(x)`
   - `psi1.2(x)`
   - `psi1.3(x)`
   - `psi1.4(x)`
   - `phi1(x)`

   Estas funciones están disponibles para que instrucciones subsiguientes las utilicen como salida directamente con `WRITE_MESH`, como parte de otras expresiones intermedias, etc.
4. Si el problema fuese de criticidad, entonces esta instrucción también pondría el valor del factor de multiplicación efectivo k_{eff} en una variable llamada `keff`.

- La siguiente línea define una función auxiliar de la variable espacial x a partir de un archivo de columnas de datos. Este archivo contiene una solución de referencia del problema de Reed. La función `ref(x)` puede ser evaluada en cualquier punto x utilizando una interpolación monotónica cúbica de tipo `steffen` [59].

- La última línea imprime en la salida estándar una representación ASCII (por defecto utilizando el formato `%g` de la instrucción `printf()` de C) del error L_2 cometido por la solución calculada con FeenoX con respecto a la solución de referencia, es decir

$$\frac{1}{8} \cdot \sqrt{\int_0^8 \left[\text{ref}(x) - \phi_1(x)\right]^2}$$

La ejecución de FeenoX con este archivo de entrada para S_2, S_4, S_6 y S_8 da como resultado

```
$ feenox reed.fee 2
0.0505655
$ feenox reed.fee 4
0.0143718
$ feenox reed.fee 6
0.010242
$ feenox reed.fee 8
0.0102363
$
```

4.1.4. Puntos de entrada

La compilación del código fuente usa el procedimiento recomendado por GNU donde el script `configure` genera los archivos de *make*[29] según

a. la arquitectura del hardware (Intel, ARM, etc.)
b. el sistema operativo (GNU/Linux, otras variantes, etc.)
c. las dependencias disponibles (MPI, PETSc, SLEPc, GSL, etc.)

A su vez, para generar este script `configure` se suele utilizar el conjunto de herramientas conocidas como Autotools. Estas herramientas generan, a partir de un conjunto de definiciones reducidas dadas en el lenguaje de macros M4, no sólo el script `configure` sino también otros archivos relacionados al proceso de compilación tales como los templates para los makefiles. Estas definiciones reducidas (que justamente definen las arquitecturas y sistemas operativos soportados, las dependencias, etc.) usualmente se dan en un archivo de texto llamado `configure.ac` y los templates que indican dónde están los archivos fuente que se deben compilar en archivos llamados `Makefile.am` ubicados en uno o más subdirectorios. Éstos últimos se referencian desde `configure.ac` de forma tal que Autoconf y Automake trabajen en conjunto para generar el script `configure`, que forma parte de la distribución del código fuente de forma tal que un usuario arbitrario pueda ejecutarlo y luego compilar el código con el comando `make`, que lee el `Makefile` generado por `configure`.

Para poder implementar la idea de extensibilidad según la cual FeenoX podría resolver diferentes ecuaciones en derivadas parciales, le damos una vuelta más de tuerca a esta idea de generar archivos a partir de scripts. Para ello empleamos la idea de *bootstrapping* (figura 4.3), en la cual el archivo `configure.ac` y/o las plantillas `Makefile.am` son generadas a partir de un script llamado `autogen.sh` (algunos autores prefieren llamarlo `bootstrap`).

Figura 4.3.: El concepto de `bootstrap` (también llamado `autogen.sh`).

Este script `autogen.sh` detecta qué subdirectorios hay dentro del directorio `src/pdes` y, para cada uno de ellos, agrega unas líneas a un archivo fuente llamado `src/pdes/parse.c` que hace apuntar un cierto apuntador a función a una de las funciones definidas dentro del subdirectorio. En forma resumida,

[29] Del inglés *make files*.

```
for pde in *; do
 if [ -d ${pde} ]; then
  if [ ${first} -eq 0 ]; then
    echo -n "  } else " >> parse.c
  else
    echo -n "  " >> parse.c
  fi
  cat << EOF >> parse.c
if (strcasecmp(token, "${pde}") == 0) {
  feenox.pde.parse_problem = feenox_problem_parse_problem_${pde};

EOF

  first=0
 fi
done
```

Esto generaría el bloque de `if`s feo que ya mencionamos en `parse.c`:

```
if (strcasecmp(token, "laplace") == 0) {
  feenox.pde.parse_problem = feenox_problem_parse_problem_laplace;
} else if (strcasecmp(token, "mechanical") == 0) {
  feenox.pde.parse_problem = feenox_problem_parse_problem_mechanical;
} else if (strcasecmp(token, "modal") == 0) {
  feenox.pde.parse_problem = feenox_problem_parse_problem_modal;
} else if (strcasecmp(token, "neutron_diffusion") == 0) {
  feenox.pde.parse_problem = feenox_problem_parse_problem_neutron_diffusion;
} else if (strcasecmp(token, "neutron_sn") == 0) {
  feenox.pde.parse_problem = feenox_problem_parse_problem_neutron_sn;
} else if (strcasecmp(token, "thermal") == 0) {
  feenox.pde.parse_problem = feenox_problem_parse_problem_thermal;
} else {
  feenox_push_error_message("unknown problem type '%s'", token);
  return FEENOX_ERROR;
}
```

Estas instrucciones son llamadas desde el parser general luego de haber leído la definición `PROBLEM`. Por ejemplo, si en el archivo de entrada se encuentra esta línea

```
PROBLEM laplace
```

entonces el apuntador a función `feenox.pde.parse_problem` declarado en `feenox.h` como

```
int (*parse_problem)(const char *token);
```

apuntaría a la función `feenox_problem_parse_problem_laplace()` declarada en `src/pdes/laplace/methods.h` y definida en `src/pdes/laplace/parser.c`. De hecho, esta función es llamada con el parámetro `token` conteniendo cada una de las palabras que está a continuación del nombre del problema que el parser general no entienda. En particular, para la línea

```
PROBLEM neutron_sn DIM 3 GROUPS 2 SN 8
```

lo que sucede en tiempo de parseo es

1. La palabra clave primaria `PROBLEM` es leída (y entendida) por el parser general.

2. El argumento `neutron_sn` es leído por el bloque generado por `autogen.sh`. El apuntador a la función global `feenox.pde.parse_problem` se hace apuntar entonces a la función provista en `src/pdes/neutron_sn` llamada `feenox_problem_parse_problem_neutron_sn()`.

3. La palabra clave secundaria `DIM` es leída por el parser general. Como es una palabra clave secundaria asociada a la primaria `PROBLEM` y el parser general la entiende (ya que el framework tiene que saber la dimensión espacial de la ecuación diferencial en derivadas parciales que tiene que resolver, entonces la *parsea*). Seguidamente, lee el siguiente argumento `3` (que puede ser una expresión como comentamos más adelante) y sabe que tiene que resolver una ecuación diferencial sobre tres dimensiones espaciales. Esto implica, por ejemplo, definir que las propiedades de los materiales y las soluciones serán funciones de tres argumentos: x, y y z.

4. La palabra clave secundaria `GROUPS` es leída por el parser general. Como no es una palabra clave secundaria asociada a la primara `PROBLEM`, entonces se llama a `feenox.pde.parse_problem` (que apunta a `feenox_problem_parse_problem_neutron_sn()`) con el argumento `token` apuntando a `GROUPS`.

5. Como el parser particular dentro de `src/pdes/neutron_sn` sí entiende que la palabra clave `GROUPS` define la cantidad de grupos de energía, lee el siguiente token `2` (que también puede ser una expresión, ver sección 4.2), lo entiende como tal y lo almacena en la estructura de datos particular de las rutinas correspondientes a `neutron_sn`.

6. El control vuelve al parser principal que lee la siguiente palabra clave secundaria `SN`. Como tampoco la entiende, vuelve a llamar al parser particular que entiende que debe utilizar las direcciones y pesos de S_8.

7. Una vez más el control vuelve al parser principal, que llega al final de la línea. En este momento, vuelve a llamar al parser específico `feenox_problem_parse_problem_neutron_sn()` pero pasando `NULL` como argumento. En este punto, se considera que el parser específico ya tiene toda la información necesaria para inicializar (al menos una parte) de sus estructuras internas y de las variables o funciones que deben estar disponibles para otras palabras claves genéricas. Por ejemplo, si el problema es neutrónico entonces inmediatamente después de haber parseado completamente la línea `PROBLEM` debe definirse la variable `keff` y las funciones con los flujos escalares (y angulares si correspondiere) de forma tal que las siguientes líneas, que serán interpretadas por el parser genérico, entiendan que `keff` es una variable y que `phi1(x,y,z)`, `psi1.1(x,y,z)` y `psi2.8(x,y,z)` son expresiones válidas:

```
PRINT "keff = " keff
PRINT " rho = " (1-keff)/keff
PRINT psi1.1(0,0,0) psi8.2(0,0,0)
profile(x) = phi1(x,x,0)
PRINT_FUNCTION profile phi1(x,0,0) MIN 0 MAX 20 NSTEPS 100
```

Dentro de las inicializaciones en tiempo de parseo, cada implementación específica debe resolver el resto de los apuntadores a función que definen los puntos de entrada específicos que el framework principal necesita llamar para

i. parsear partes específicas del archivo de entrada

   a. condiciones de contorno

   b. la palabra clave `WRITE_RESULTS` que escribe "automáticamente" los resultados en un archivo de post-procesamiento en formato `.msh` o `.vtk`. Esto es necesario ya que las rutinas que escriben los resultados son parte del framework general pero dependiendo de la PDE a resolver e incluso de los detalles de la PDE (por ejemplo la cantidad de grupos de energía en un problema neutrónico o la cantidad de modos calculadas en un problema de análisis de modos naturales de oscilación mecánicos).

ii. inicializar estructuras internas

iii. reservar la memoria virtual necesaria con una llamada al sistema operativo[30] y construir las matrices y los vectores globales

iv. resolver las ecuaciones discretizadas con PETSc (o SLEPc) según el tipo de problema resultante:

   a. problema lineal en estado estacionario $\mathsf{K} \cdot \mathbf{u} = \mathbf{b}$ (PETSc KSP)
   b. problema generalizado de autovalores $\mathsf{K} \cdot \mathbf{u} = \lambda \cdot \mathsf{M} \cdot \mathbf{u}$ (SLEPc EPS)
   c. problema no lineal en estado estacionario $\mathbf{F}(\mathbf{u}) = 0$ (PETSc SNES)
   d. problema transitorio $\mathbf{G}(\mathbf{u}, \dot{\mathbf{u}}, t) = 0$ (PETSc TS)

v. calcular campos secundarios a partir de los primarios (ver sección 4.1.4.5), por ejemplo

   - flujos escalares a partir de flujos angulares
   - corrientes a partir de flujos neutrónicos
   - flujos de calor a partir de temperaturas
   - tensiones a partir de deformaciones
   - etc.

```
// parse
int (*parse_problem)(const char *token);
int (*parse_write_results)(mesh_write_t *mesh_write, const char *token);
int (*parse_bc)(bc_data_t *bc_data, const char *lhs, char *rhs);

// init
int (*init_before_run)(void);
int (*setup_pc)(PC pc);
int (*setup_ksp)(KSP ksp);
int (*setup_eps)(EPS eps);
int (*setup_ts)(TS ksp);

// build
int (*element_build_volumetric)(element_t *e);
int (*element_build_volumetric_at_gauss)(element_t *e, unsigned int q);

// solve
int (*solve)(void);

// post
int (*solve_post)(void);
int (*gradient_fill)(void);
int (*gradient_nodal_properties)(element_t *e, mesh_t *mesh);
int (*gradient_alloc_nodal_fluxes)(node_t *node);
int (*gradient_add_elemental_contribution_to_node)(node_t *node, element_t *e, unsigned int
    j, double rel_weight);
int (*gradient_fill_fluxes)(mesh_t *mesh, size_t j_global);
```

[30] Del inglés *allocate with a system call*.

4.1.4.1. Parseo

Cuando se termina la línea de `PROBLEM`, el parser general llama a `parse_problem(NULL)` que debe

1. terminar de rellenar los apuntadores específicos

```
// virtual methods
feenox.pde.parse_bc = feenox_problem_bc_parse_neutron_diffusion;
feenox.pde.parse_write_results = feenox_problem_parse_write_post_neutron_diffusion;

feenox.pde.init_before_run = feenox_problem_init_runtime_neutron_diffusion;

feenox.pde.setup_eps = feenox_problem_setup_eps_neutron_diffusion;
feenox.pde.setup_ksp = feenox_problem_setup_ksp_neutron_diffusion;
feenox.pde.setup_pc = feenox_problem_setup_pc_neutron_diffusion;

feenox.pde.element_build_volumetric = feenox_problem_build_volumetric_neutron_diffusion;
feenox.pde.element_build_volumetric_at_gauss =  ↩
    feenox_problem_build_volumetric_gauss_point_neutron_diffusion;

feenox.pde.solve_post = feenox_problem_solve_post_neutron_diffusion;
```

2. inicializar lo que necesita el parser para poder continuar leyendo el problema específico, incluyendo

   - la definición de variables especiales (por ejemplo los flujos escalares `phi_1`, `phi_2`, etc. y angulares `psi_1.1`, `psi_1.2`, ..., `psi_2.12` y las variables `keff` y `sn_alpha`) para que estén disponibles para el parser algebraico (ver sección 4.2)

```
// NOTE: it is more natural to put first the group and then the direction
//       while in the doc we use $\psi_{mg}$
for (unsigned int g = 0; g < neutron_sn.groups; g++) {
  for (unsigned int m = 0; m < neutron_sn.directions; m++) {
    feenox_check_minusone(asprintf(&feenox.pde.unknown_name[sn_dof_index(m,g)],  ↩
        "psi%u.%u", g+1, m+1));
  }
}

// the scalar fluxes phi
feenox_check_alloc(neutron_sn.phi = calloc(neutron_sn.groups, sizeof(function_t *)));
for (unsigned int g = 0; g < neutron_sn.groups; g++) {
  char *name = NULL;
  feenox_check_minusone(asprintf(&name, "phi%u", g+1));
  feenox_call(feenox_problem_define_solution_function(name, &neutron_sn.phi[g],  ↩
      FEENOX_SOLUTION_NOT_GRADIENT));
  feenox_free(name);
}

neutron_sn.keff = feenox_define_variable_get_ptr("keff");

neutron_sn.sn_alpha = feenox_define_variable_get_ptr("sn_alpha");
feenox_var_value(neutron_sn.sn_alpha) = 0.5;
```

   - la colocación de opciones por defecto (¿qué pasa si no hay keywords `GROUPS` o `SN`?)

```
// default is 1 group
if (neutron_sn.groups == 0) {
  neutron_sn.groups = 1;
}

// default is N=2
if (neutron_sn.N == 0) {
  neutron_sn.N = 2;
}
if (neutron_sn.N % 2 != 0) {
  feenox_push_error_message("number of ordinates N = %d has to be even",
      neutron_sn.N);
  return FEENOX_ERROR;
}

// stammler's eq 6.19
switch(feenox.pde.dim) {
  case 1:
    neutron_sn.directions = neutron_sn.N;
    break;
  case 2:
    neutron_sn.directions = 0.5*neutron_sn.N*(neutron_sn.N+2);
    break;
  case 3:
    neutron_sn.directions = neutron_sn.N*(neutron_sn.N+2);
    break;
}

// dofs = number of directions * number of groups
feenox.pde.dofs =  neutron_sn.directions * neutron_sn.groups;
```

- la inicialización de direcciones $\hat{\mathbf{\Omega}}_m$ y pesos w_m para S_N

Cuando una línea contiene la palabra clave principal `BC` tal como

```
BC left  mirror
BC right vacuum
```

entonces el parser principal lee el token siguiente, `left` y `right` respectivamente, que intentará vincular con un grupo físico en la malla del problema. Los siguientes tokens, en este caso sólo uno para cada caso, son pasados al parser específico `feenox.pde.parse_bc` que sabe qué hacer con las palabras `mirror` y `vacuum`. Este parser de condiciones de contorno a su vez resuelve uno de los dos apuntadores

```
int (*set_essential)(bc_data_t *, element_t *, size_t j_global);
int (*set_natural)(bc_data_t *, element_t *, unsigned int q);
```

dentro de la estructura asociada a la condición de contorno según corresponda al tipo de condición.

4.1.4.2. Inicialización

Luego de que el parser lee completamente el archivo de entrada, FeenoX ejecuta las instrucciones en el orden adecuado, opcional y/o posiblemente siguiendo lazos de control y bucles.[31] Al llegar a la

[31] Del inglés *loop*

instrucción SOLVE_PROBLEM, llama al punto de entrada `init_before_run()` donde

1. Se leen las expresiones que definen las propiedades de los materiales, que no estaban disponibles en el momento de la primera inicialización después de leer la línea PROBLEM. Estas propiedades de materiales en general y las secciones eficaces macroscópicas en particular pueden estar dadas por

   a. variables
   b. funciones del espacio
   c. la palabra clave MATERIAL

   En este momento de la ejecución todas las secciones eficaces deben estar definidas con nombres especiales. Por ejemplo,

   - Σ_{tg} = `"Sigma_t%d"`
   - Σ_{ag} = `"Sigma_a%d"`
   - $\nu\Sigma_{fg}$ = `"nuSigma_f%d"`
   - $\Sigma_{s_0 g\to g'}$ = `"Sigma_s%d.%d"`
   - $\Sigma_{s_1 g\to g'}$ = `"Sigma_s_one%d.%d"`
   - S_g = `"S%d"`

2. Dependiendo de si hay fuentes de fisión y/o fuentes independientes se determina si hay que resolver un problema lineal o un problema de autovalores generalizado. En el primer caso se hace

   ```
   feenox.pde.math_type = math_type_linear;
   feenox.pde.solve     = feenox_problem_solve_petsc_linear;
   feenox.pde.has_mass  = 0;
   feenox.pde.has_rhs   = 1;
   ```

   y en el segundo

   ```
   feenox.pde.math_type = math_type_eigen;
   feenox.pde.solve     = feenox_problem_solve_slepc_eigen;
   feenox.pde.has_mass  = 1;
   feenox.pde.has_rhs   = 0;
   ```

Luego de esta segunda inicialización, se llama al apuntador `feenox.pde.solve` que para neutrónica es o bien

a. `feenox_problem_solve_petsc_linear()` que construye K y $\mathbf{b}$ y resuelve $\mathsf{K}\cdot\mathbf{u}=\mathbf{b}$, o
b. `feenox_problem_solve_slepc_eigen()` que construye K y M y resuelve $\mathsf{K}\cdot\mathbf{u}=\lambda\cdot\mathsf{M}\cdot\mathbf{u}$.

Antes de construir y resolver las ecuaciones, se llama a su vez a los apuntadores a función que correspondan

- `feenox_pde.setup_pc(PC pc)`
- `feenox.pde.setup_ksp(KSP kps)`
- `feenox.pde.setup_eps(EPS eps)`

donde cada problema particular configura el pre-condicionador, el solver lineal y el solver de autovalores en caso de que el usuario no haya elegido algoritmos explícitamente en el archivo de entrada. Si el operador diferencial es elíptico y simétrico (por ejemplo para conducción de calor o elasticidad lineal) tal vez convenga usar por defecto un solver iterativo basado en gradientes conjugados pre-condicionado con multi-grilla geométrica-algebraica[32] [6]. En cambio para un operador hiperbólico no simétrico (por ejemplo S_N multigrupo) es necesario un solver más robusto como por ejemplo LU.

4.1.4.3. Construcción

Para construir las matrices globales `K` y/o `M` y/o el vector global $\mathbf{b}$, el framework general hace un bucle sobre todos los elementos volumétricos (locales a cada proceso) y, para cada uno de ellos, primero llama al punto de entrada `feenox.pde.element_build_volumetric()` que toma un apuntador al elemento como único argumento. Cada elemento es una estructura tipo `element_t` definida en `feenox.h` como

```
struct element_t {
  size_t index;
  size_t tag;

  double quality;
  double volume;
  double area;
  double size;
  double gradient_weight;   // this weight is used to average the contribution of this element to nodal gradients
  double *w;                // weights of the gauss points time determinant of the jacobian
  double **x;               // coordinates fo the gauss points
  double *normal;           // outward normal direction (only for 2d elements)

  // matrix with the coordinates (to compute the jacobian)
  gsl_matrix *C;

  element_type_t *type;                 // pointer to the element type
  physical_group_t *physical_group;     // pointer to the physical group this element belongs to
  node_t **node;                        // pointer to the nodes, node[j] points to the j-th local node
  cell_t *cell;                         // pointer to the associated cell (only for FVM)
};
```

Luego el framework general hace un sub-bucle sobre el índice q de los puntos de Gauss y llama a `feenox.pde.element_build_volumetric_at_gauss()` que toma el apuntador al elemento y el índice q como argumentos:

```
// if the specific pde wants, it can loop over gauss point in the call
// or it can just do some things that are per-element only and then loop below
if (feenox.pde.element_build_volumetric) {
  feenox_call(feenox.pde.element_build_volumetric(this));
}

// or, if there's an entry point for gauss points, then we do the loop here
if (feenox.pde.element_build_volumetric_at_gauss != NULL) {
  int Q = this->type->gauss[feenox.pde.mesh->integration].Q;
```

[32] Del inglés *geometric algebraic multi-grid*.

```
  for (unsigned int q = 0; q < Q; q++) {
    feenox_call(feenox.pde.element_build_volumetric_at_gauss(this, q));
  }
}
```

Estos dos métodos son responsables de devolver los objetos elementales volumétricos K_i, M_i como matrices densas en formato de la GNU Scientific Library `gsl_matrix` y el vector elemental $\mathbf{b}_i$ como `gsl_vector` almacenados en los apuntadores globales dentro de la estructura `feenox.fem`:

```
// elemental (local) objects
gsl_matrix *Ki;               // elementary stiffness matrix
gsl_matrix *Mi;               // elementary mass matrix
gsl_matrix *JKi;              // elementary jacobian for stiffness matrix
gsl_matrix *Jbi;              // elementary jacobian for RHS vector
gsl_vector *bi;               // elementary right-hand side vector
```

Observación. Las matrices jacobianas son necesarias para problemas no lineales resueltos con SNES.

Para el caso de difusión de neutrones, las dos funciones que construyen los objetos elementales son

```
int feenox_problem_build_volumetric_neutron_diffusion(element_t *e) {
  if (neutron_diffusion.space_XS == 0) {
    feenox_call(feenox_problem_neutron_diffusion_eval_XS(feenox_fem_get_material(e), NULL));
  }
  return FEENOX_OK;
}

int feenox_problem_build_volumetric_gauss_point_neutron_diffusion(element_t *e, unsigned int
    q) {
  if (neutron_diffusion.space_XS != 0) {
    double *x = feenox_fem_compute_x_at_gauss(e, q, feenox.pde.mesh->integration);
    feenox_call(feenox_problem_neutron_diffusion_eval_XS(feenox_fem_get_material(e), x));
  }

  // elemental stiffness for the diffusion term B'*D*B
  double wdet = feenox_fem_compute_w_det_at_gauss(e, q, feenox.pde.mesh->integration);
  gsl_matrix *B = feenox_fem_compute_B_G_at_gauss(e, q, feenox.pde.mesh->integration);
  feenox_call(feenox_blas_BtCB(B, neutron_diffusion.D_G, neutron_diffusion.DB, wdet,
      neutron_diffusion.Li));

  // elemental scattering H'*A*H
  gsl_matrix *H = feenox_fem_compute_H_Gc_at_gauss(e, q, feenox.pde.mesh->integration);
  feenox_call(feenox_blas_BtCB(H, neutron_diffusion.R, neutron_diffusion.RH, wdet,
      neutron_diffusion.Ai));

  // elemental fission matrix
  if (neutron_diffusion.has_fission) {
    feenox_call(feenox_blas_BtCB(H, neutron_diffusion.X, neutron_diffusion.XH, wdet,
        neutron_diffusion.Fi));
  }

  if (neutron_diffusion.has_sources) {
    feenox_call(feenox_blas_Atb_accum(H, neutron_diffusion.s, wdet, feenox.fem.bi));
  }

  // for source-driven problems
  //   Ki = Li + Ai - Xi
```

```
  // for criticallity problems
  //   Ki = Li + Ai
  //   Mi = Xi
  feenox_call(gsl_matrix_add(neutron_diffusion.Li, neutron_diffusion.Ai));
  if (neutron_diffusion.has_fission) {
    if (neutron_diffusion.has_sources) {
      feenox_call(gsl_matrix_scale(neutron_diffusion.Fi, -1.0));
      feenox_call(gsl_matrix_add(neutron_diffusion.Li, neutron_diffusion.Fi));
    } else {
      feenox_call(gsl_matrix_add(feenox.fem.Mi, neutron_diffusion.Fi));
    }
  }
  feenox_call(gsl_matrix_add(feenox.fem.Ki, neutron_diffusion.Li));

  return FEENOX_OK;
}
```

Luego de hacer el bucle sobre cada punto de Gauss q de cada elemento volumétrico e_i, el framework general se encarga de ensamblar los objetos globales K, M y/o $\mathbf{b}$ en formato PETSc a partir de los objetos locales K_i, M_i y/o $\mathbf{b}_i$ en formato GSL. En forma análoga, el framework hace otro bucle sobre los elementos superficiales que contienen condiciones de borde naturales y llama al apuntador a función `set_natural()` que toma como argumentos

1. un apuntador a una estructura `bc_data_t` definida como

```
struct bc_data_t {
  char *string;

  enum {
    bc_type_math_undefined,
    bc_type_math_dirichlet,
    bc_type_math_neumann,
    bc_type_math_robin,
    bc_type_math_multifreedom
  } type_math;

  int type_phys;      // problem-based flag that tells which type of BC this is

  // boolean flags
  int space_dependent;
  int nonlinear;
  int disabled;
  int fills_matrix;

  unsigned int dof;  // -1 means "all" dofs
  expr_t expr;
  expr_t condition;  // if it is not null the BC only applies if this evaluates to non-zero

  int (*set_essential)(bc_data_t *, element_t *, size_t j_global);
  int (*set_natural)(bc_data_t *, element_t *, unsigned int q);

  bc_data_t *prev, *next;   // doubly-linked list in ech bc_t
};
```

   que es "rellenada" por el parser específico de condiciones de contorno,

2. un apuntador al elemento

3. el índice q del punto de Gauss

Por ejemplo, la condición de contorno natural tipo `vacuum` de difusión de neutrones es

```
int feenox_problem_bc_set_neutron_diffusion_vacuum(bc_data_t *this, element_t *e, unsigned int q) {

  feenox_fem_compute_x_at_gauss_if_needed_and_update_var(e, q, feenox.pde.mesh->integration, this->space_dependent);
  double coeff = (this->expr.items != NULL) ? feenox_expression_eval(&this->expr) : 0.5;

  double wdet = feenox_fem_compute_w_det_at_gauss(e, q, feenox.pde.mesh->integration);
  gsl_matrix *H = feenox_fem_compute_H_Gc_at_gauss(e, q, feenox.pde.mesh->integration);
  feenox_call(feenox_blas_BtB_accum(H, wdet*coeff, feenox.fem.Ki));

  return FEENOX_OK;
}
```

Observación. La condición de contorno tipo `mirror` en difusión, tal como en la ecuación de Laplace o en conducción de calor, consiste en "no hacer nada".

Las condiciones de contorno esenciales se ponen en el problema luego de haber ensamblado la matriz global A para transformarla en la matriz de rigidez K según el procedimiento discutido en la sección 3.4.1.5. Para ello debemos hacer un bucle sobre los nodos, bien con el algoritmo 3 o con el algoritmo 4, y poner un uno en la diagonal de la matriz de rigidez y el valor de la condición de contorno en el nodo en el vector del miembro derecho, en la fila correspondiente al grado de libertad global.

Observación. En un problema de autovalores, sólo es posible poner condiciones de contorno homogéneas. En este caso, se puede

a. poner un uno en la diagonal de K y un cero en la diagonal de M, ó
b. poner un uno en la diagonal de M y un cero en la diagonal de K

Cuál de las dos opciones es la más conveniente depende del algoritmo seleccionado para la transformación espectral del problema de autovalores a resolver.

La forma de implementar esto en FeenoX para una PDE arbitraria es que para cada nodo que contiene una condición de Dirichlet, el framework llama al apuntador a función `set_essential()` que toma como argumentos

1. un apuntador a la estructura `bc_data_t`
2. un apuntador al elemento (estructura `element_t`)
3. el índice j global del nodo

Para difusión, la condición `null` se implementa sencillamente como

```
int feenox_problem_bc_set_neutron_diffusion_null(bc_data_t *this, element_t *e, size_t j_global) {
  for (unsigned int g = 0; g < feenox.pde.dofs; g++) {
    feenox_call(feenox_problem_dirichlet_add(feenox.pde.mesh->node[j_global].index_dof[g], 0));
  }
  return FEENOX_OK;
}
```

En S_N, la condición `vacuum` es ligeramente más compleja ya que debemos verificar que la dirección sea entrante:

```
int feenox_problem_bc_set_neutron_sn_vacuum(bc_data_t *this, element_t *e, size_t j_global) {

  double outward_normal[3];
  feenox_call(feenox_mesh_compute_outward_normal(e, outward_normal));
  for (unsigned m = 0; m < neutron_sn.directions; m++) {
    if (feenox_mesh_dot(neutron_sn.Omega[m], outward_normal) < 0) {
      // if the direction is inward set it to zero
      for (unsigned int g = 0; g < neutron_sn.groups; g++) {
        feenox_call(feenox_problem_dirichlet_add(
         feenox.pde.mesh->node[j_global].index_dof[sn_dof_index(m,g)], 0)
        );
      }
    }
  }

  return FEENOX_OK;
}
```

La condición de contorno `mirror` es más compleja aún porque implica condiciones multi-libertad[33], que deben ser puestas en la matriz de rigidez usando

1. Eliminación directa
2. Método de penalidad
3. Multiplicadores de Lagrange

En su versión actual, FeenoX utiliza el método de penalidad [18]. El framework provee una llamada genérica `feenox_problem_multifreedom_add()` donde las rutinas particulares deben "registrar" sus condiciones multi-libertad:

```
int feenox_problem_bc_set_neutron_sn_mirror(bc_data_t *this, element_t *e, size_t j_global) {

  double outward_normal[3];
  double reflected[3];
  double Omega_dot_outward = 0;
  double eps = feenox_var_value(feenox.mesh.vars.eps);

  feenox_call(feenox_mesh_compute_outward_normal(e, outward_normal));
  for (unsigned m = 0; m < neutron_sn.directions; m++) {
    if ((Omega_dot_outward = feenox_mesh_dot(neutron_sn.Omega[m], outward_normal)) < 0) {
      // if the direction is inward then we have to reflect it
      // if Omega is the incident direction with respect to the outward normal then
      // reflected = Omega - 2*(Omega dot outward_normal) * outward_normal
      for (int d = 0; d < 3; d++) {
        reflected[d] = neutron_sn.Omega[m][d] - 2*Omega_dot_outward * outward_normal[d];
      }

      unsigned int m_prime = 0;
      for (m_prime = 0; m_prime < neutron_sn.directions; m_prime++) {
        if (fabs(reflected[0] - neutron_sn.Omega[m_prime][0]) < eps &&
            fabs(reflected[1] - neutron_sn.Omega[m_prime][1]) < eps &&
            fabs(reflected[2] - neutron_sn.Omega[m_prime][2]) < eps) {
          break;
        }
```

[33] Del inglés *multi-freedom*.

```
      }

      if (m_prime == neutron_sn.directions) {
        feenox_push_error_message("cannot find a reflected direction for m=%d (out of %d in S%d) for node %d", m, neutron_sn.directions, neutron_sn.N, feenox.pde.mesh->node[j_global].tag);
        return FEENOX_ERROR;
      }

      double *coefficients = NULL;
      feenox_check_alloc(coefficients = calloc(feenox.pde.dofs, sizeof(double)));

      for (unsigned int g = 0; g < neutron_sn.groups; g++) {
        coefficients[sn_dof_index(m,g)] = -1;
        coefficients[sn_dof_index(m_prime,g)] = +1;
      }
      feenox_call(feenox_problem_multifreedom_add(j_global, coefficients));
      feenox_free(coefficients);
    }
  }

  return FEENOX_OK;
}
```

Éstas son luego multiplicadas por un factor de penalidad y sumadas a la matriz de rigidez por el framework general.

4.1.4.4. Solución

Una vez construidos los objetos globales `K` y/o `M` y/o **b**, el framework llama a `feenox.pde.solve()` que puede apuntar a

1. `feenox_problem_solve_petsc_linear()`
2. `feenox_problem_solve_petsc_nonlinear()`
3. `feenox_problem_solve_slepc_eigen()`
4. `feenox_problem_solve_petsc_transient()`

según el inicializador particular de la PDE haya elegido. Por ejemplo, si la variable `end_time` es diferente de cero, entonces en `thermal` se llama a `trasient()`. De otra manera, si la conductividad k depende de la temperatura T, se llama a `nonlinear()` y si no depende de T, se llama a `linear()`. En el caso de neutrónica, tanto difusión como S_N, la lógica depende de si existen fuentes independientes o no:

```
feenox.pde.math_type = neutron_diffusion.has_sources ? math_type_linear : math_type_eigen;
feenox.pde.solve     = neutron_diffusion.has_sources ? feenox_problem_solve_petsc_linear : feenox_problem_solve_slepc_eigen;

feenox.pde.has_stiffness = 1;
feenox.pde.has_mass = !neutron_diffusion.has_sources;
feenox.pde.has_rhs  =  neutron_diffusion.has_sources;
```

4.1.4.5. Post-procesamiento

Luego de resolver el problema discretizado y de encontrar el vector global solución **u**, debemos dejar disponibles la solución para que las instrucciones que siguen a `SOLVE_PROBLEM`, tales como las palabras clave `PRINT` o `WRITE_RESULTS`, puedan operar con ellas.

Definición 4.2 (campos principales). Las funciones del espacio que son solución de la ecuación a resolver y cuyos valores nodales provienen de los elementos del vector solución **u** se llaman *campos principales*.

Definición 4.3 (campos secundarios). Las funciones del espacio que dan información sobre la solución del problema cuyos valores nodales provienen de operar algebraica, diferencial o integralmente sobre el vector solución **u** se llaman *campos secundarios*.

Problema	Campo principal	Campos secundarios
Conducción de calor	Temperatura	Flujos de calor
Elasticidad (formulación u)	Desplazamientos	Tensiones y deformaciones
Difusión de neutrones	Flujos escalares	Corrientes
Ordenadas discretas	Flujos angulares	Flujos escalares y corrientes

Tabla 4.1.: Campos principales y secundarios.

La tabla 4.1 lista los campos principales y secundarios de algunos tipos de problemas físicos. Los campos principales son "rellenados" por el framework general mientras que los campos secundarios necesitan más puntos de entrada específicos para poder ser definidos.

En efecto, el framework sabe cómo "rellenar" las funciones solución a partir de **u** mediante los índices que mapean los grados de libertad de cada nodo espacial con los índices globales. Para ello, como ya vimos, cada inicializador debe "registrar" la cantidad y el nombre de las soluciones según la cantidad de grados de libertad por nodo dado en `feenox.pde.dofs`.

Por ejemplo, para un problema escalar como `thermal`, hay un único grado de libertad por nodo por lo que debe haber una única función solución de la ecuación que el inicializador "registró" en el framework como

```
// thermal is a scalar problem
feenox.pde.dofs = 1;
feenox_check_alloc(feenox.pde.unknown_name = calloc(feenox.pde.dofs, sizeof(char *)));
feenox_check_alloc(feenox.pde.unknown_name[0] = strdup("T"));
```

En difusión de neutrones, la cantidad de grados de libertad por nodo es la cantidad G de grupos de energía, leído por el parser específico en la palabra clave `GROUPS`. Las funciones solución son `phi1`, `phi2`, etc:

```
// default is 1 group
if (neutron_diffusion.groups == 0) {
  neutron_diffusion.groups = 1;
}
// dofs = number of groups
feenox.pde.dofs = neutron_diffusion.groups;

feenox_check_alloc(feenox.pde.unknown_name = calloc(neutron_diffusion.groups, sizeof(char *)));
for (unsigned int g = 0; g < neutron_diffusion.groups; g++) {
  feenox_check_minusone(asprintf(&feenox.pde.unknown_name[g], "phi%u", g+1));
}
```

En S_N, la cantidad de grados de libertad por nodo es el producto entre M y G. Las funciones, como mostramos en la sección 4.1.4.1, son `psi1.1`, `psi2.1`, etc. donde el primer índice es g y el segundo es m.

Observación. En la notación matemática de los capítulos 2 y 3 es más natural escribir ψ_{mg} para la dirección m y el grupo g. Pero en el archivo ASCII de entrada de FeenoX es más natural escribir `psig.m` para la dirección `m` y el grupo `g`.

4.2. Expresiones algebraicas

Una característica distintiva de FeenoX es que en cada lugar del archivo de entrada donde se espere un valor numérico, desde la cantidad de grupos de energía después de la palabra clave `GROUPS` hasta las propiedades de los materiales, es posible escribir una expresión algebraica. Por ejemplo

```
PROBLEM neutron_diffusion DIMENSIONS 1+2 GROUPS sqrt(4)
MATERIAL fuel nuSigma_f1=1+T(x,y,z)  nuSigma_f2=10-1e-2*(T(x,y,z)-T0)^2
```

Esto obedece a una de las primeras decisiones de diseño del código. Parafraseando la idea de Unix "todo es un archivo", para FeenoX "todo es una expresión". La explicación se basa en la historia misma de por qué en algún momento de mi vida profesional es que decidí escribir la primera versión de este código, cuya tercera implementación es FeenoX Apéndice D. Luego de la tesis de grado [60] y de la de maestría [61], en el año 2009 busqué en StackOverflow cómo implementar un parser PEMDAS (*Parenthesis Exponents Multiplication Division Addition Subtraction*) en C. Esa primera implementación solamente soportaba constantes numéricas, por lo que luego agregué la posibilidad de incorporar variables y funciones matemáticas estándar (trigonométricas, exponenciales, etc.) cuyos argumentos son, a su vez, expresiones algebraicas. Finalmente funciones definidas por el usuario (sección 4.3) e incluso funcionales que devuelven un número real a partir de una expresión (implementadas con la GNU Scientific Library) como por ejemplo

- derivación

  ```
  VAR t'
  f'(t) = derivative(f(t'),t',t)
  ```

- integración

```
# this integral is equal to 22/7-pi
piapprox = 22/7-integral((x^4*(1-x)^4)/(1+x^2), x, 0, 1)
```

- sumatoria

```
# the abraham sharp sum (twenty-one terms)
piapprox = sum(2*(-1)^i * 3^(1/2-i)/(2*i+1), i, 0, 20)
```

- búsqueda de extremos en un dimensión

```
PRINT %.7f func_min(cos(x)+1,x,0,6)
```

- búsqueda de raíces en una dimensión

```
VECTOR kl[5]
kl[i] = root(cosh(t)*cos(t)+1, t, 3*i-2,3*i+1)
```

La forma en la que FeenoX maneja expresiones es la siguiente:

1. El parser toma una cadena de caracteres y la analiza (*parsea*, por eso es un *parser*) para generar un árbol de sintaxis abstracto[34] que consiste en una estructura `expr_t`

```
// algebraic expression
struct expr_t {
  expr_item_t *items;
  double value;
  char *string;     // just in case we keep the original string

  // lists of which variables and functions this expression depends on
  var_ll_t *variables;
  function_ll_t *functions;

  expr_t *next;
};
```

   que tiene una lista simplemente vinculada[35] de estructuras `expr_item_t`

```
// individual item (factor) of an algebraic expression
struct expr_item_t {
  size_t n_chars;
  int type;           // defines #EXPR_ because we need to operate with masks

  size_t level;       // hierarchical level
  size_t tmp_level;   // for partial sums

  size_t oper;        // number of the operator if applicable
  double constant;    // value of the numerical constant if applicable
  double value;       // current value

  // vector with (optional) auxiliary stuff (last value, integral accumulator, rng, etc)
  double *aux;
```

[34] Del inglés *abstract syntax tree*.
[35] Del inglés *single-linked list*.

```
  builtin_function_t *builtin_function;
  builtin_vectorfunction_t *builtin_vectorfunction;
  builtin_functional_t *builtin_functional;

  var_t *variable;
  vector_t *vector;
  matrix_t *matrix;
  function_t *function;

  vector_t **vector_arg;

  var_t *functional_var_arg;

  // algebraic expression of the arguments of the function
  expr_t *arg;

  // lists of which variables and functions this item (and its daughters)
  var_ll_t *variables;
  function_ll_t *functions;


  expr_item_t *next;
};
```

que pueden ser

a. un operador algebraico (`+`, `-`, `*`, `/` o `^`)
b. una constante numérica (`12.34` o `1.234e1`)
c. una variable (`t` o `x`)
d. un elemento de un vector (`v[1]` o `p[1+i]`)
e. un elemento de una matriz (`A(1,1)` o `B(1+i,1+j)`)
f. una función interna como por ejemplo (`sin(w*pi*t)` o `exp(-x)` o `atan2(y,x)`)
g. un funcional interno como por ejemplo (`integral(x^2,x,0,1)` o `root(cos(x)-x,x,0,1)`)
h. una función definida por el usuario (`f(x,y,z)` o `g(t)`)

2. Cada vez que se necesita el valor numérico de la expresión, se recorre la lista `items` evaluando cada uno de los factores según su orden de procedencia jerárquica PEMDAS.

Observación. Algunas expresiones como por ejemplo la cantidad de grupos de energía deben poder evaluarse en tiempo de parseo. Si bien FeenoX permite introducir expresiones en el archivo de entrada, éstas no deben depender de variables o de funciones ya que éstas necesitan estar en modo ejecución para tener valores diferentes de cero (tal como las `constexpr` de C++).

Observación. Los índices de elementos de vectores o matrices y los argumentos de funciones y funcionales son expresiones, que deben ser evaluadas en forma recursiva al recorrer la lista de ítems de una expresión base.

Observación. Los argumentos en la línea de comando `$1`, `$2`, etc. se expanden como si fuesen cadenas antes de parsear la expresión.

Esta funcionalidad, entre otras cosas, permite comparar resultados numéricos con resultados analíticos. Como muchas veces estas soluciones analíticas están dadas por series de potencias, el funcional `sum()` es muy útil para realizar esta comparación. Por ejemplo, en conducción de calor transitoria:

```
# example of a 1D heat transient problem
# from https://www.math.ubc.ca/~peirce/M257_316_2012_Lecture_20.pdf
# T(0,t) = 0       for t < 1
#          A*(t-1) for t > 1
# T(L,t) = 0
# T(x,0) = 0
READ_MESH slab-1d-0.1m.msh DIMENSIONS 1
PROBLEM thermal

end_time = 2

# unitary non-dimensional properties
k = 1
rhocp = 1
alpha = k/rhocp

# initial condition
T_0(x) = 0
# analytical solution
# example 20.2 equation 20.25
A = 1.23456789
L = 0.1
N = 100
T_a(x,t) = A*(t-1)*(1-x/L) + 2*A*L^2/(pi^3*alpha^2) *  ↩
    sum((exp(-alpha^2*(i*pi/L)^2*(t-1))-1)/i^3 * sin(i*pi*x/L), i, 1, N)

# boundary conditions
BC left  T=if(t>1,A*(t-1),0)
BC right T=0

SOLVE_PROBLEM

IF t>1 # the analytical solution overflows t<1
 PRINT %.7f t T(0.5*L) T_a(0.5*L,t) T(0.5*L)-T_a(0.5*L,t)
ENDIF
```

```
$ feenox thermal-slab-transient.fee
1.0018730       0.0006274       0.0005100       0.0001174
1.0047112       0.0021005       0.0021442       -0.0000436
1.0072771       0.0036783       0.0037210       -0.0000427
1.0097632       0.0052402       0.0052551       -0.0000150
1.0131721       0.0073607       0.0073593       0.0000014
1.0192879       0.0111393       0.0111345       0.0000048
1.0315195       0.0186881       0.0186849       0.0000032
1.0500875       0.0301491       0.0301466       0.0000025
1.0872233       0.0530725       0.0530700       0.0000025
1.1614950       0.0989193       0.0989167       0.0000026
1.3100385       0.1906127       0.1906102       0.0000026
1.6071253       0.3739997       0.3739971       0.0000026
2.0000000       0.6165149       0.6165123       0.0000026
$
```

4.3. Evaluación de funciones

Tal como las expresiones de la sección anterior, el concepto de *funciones* es central para FeenoX como oposición y negación definitiva de la idea de "tabla" para dar dependencias no triviales de las secciones eficaces con respecto a

i. temperaturas
ii. quemados
iii. concentración de venenos
iv. etc.

según lo discutido en la referencia [66] sobre la segunda versión del código.

Una función está completamente definida por

a. un nombre único f,
b. la cantidad n de argumentos que toma, y
c. por la forma de evaluar el número real $f(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}$ que corresponde a los argumentos $\mathbf{x} \in \mathbb{R}^n$.

Las funciones en FeenoX pueden ser

i. algebraicas, o
ii. definidas por puntos

4.3.1. Funciones definidas algebráicamente

En el caso de funciones algebraicas, los argumentos de la definición tienen que ser variables que luego aparecen en la expresión que define la función. El valor de la función proviene de

1. asignar a las variables de los argumentos los valores numéricos de la invocación, y luego
2. evaluar la expresión algebraica que la define.

Por ejemplo

```
f(x) = 1/2*x^2
PRINT f(1) f(2)
```

Al evaluar `f(1)` FeenoX pone el valor de la variable `x` igual a 1 y luego evalúa la expresión `1/2*x^2` dando como resultado `0.5`. En la segunda evaluación, `x` vale 2 y la función se evalúa como `2`.

Si en la expresión aparecen otras variables que no forman parte de los argumentos, la evaluación de la función dependerá del valor que tengan estas variables (que se toman como parámetros) al momento de la invocación. Por ejemplo,

```
VAR v0
v(x0,t) = x0 + v0*t
PRINT v(0,1)
v0 = 1
PRINT v(0,1)
```

en la primera evaluación obtendremos `0` y en la segunda `1`.

Observación. La figura 2.7 que ilustra los primeros armónicos esféricos fue creada por la herramienta de post-procesamiento tridimensional Paraview a partir de un archivo VTK generado por FeenoX con el siguiente archivo de entrada:

```
READ_MESH sphere.msh

Y00(x,y,z) = sqrt(1/(4*pi))

Y1m1(x,y,z) = sqrt(3/(4*pi)) * y
Y10(x,y,z)  = sqrt(3/(4*pi)) * z
Y1p1(x,y,z) = sqrt(3/(4*pi)) * x

Y2m2(x,y,z)  = sqrt(15/(4*pi)) * x*y
Y2m1(x,y,z)  = sqrt(15/(4*pi)) * y*z
Y20(x,y,z)   = sqrt(5/(16*pi)) * (-x^2-y^2+2*z^2)
Y2p1(x,y,z)  = sqrt(15/(4*pi)) * z*x
Y2p2(x,y,z)  = sqrt(15/(16*pi)) * (x^2-y^2)

WRITE_MESH harmonics.vtk Y00 Y1m1 Y10 Y1p1 Y2m2 Y2m1 Y20 Y2p1 Y2p2
```

4.3.2. Funciones definidas por puntos sin topología

Por otro lado, las funciones definidas por puntos pueden ser uni-dimensionales o multi-dimensionales. Las multi-dimensionales pueden tener o no topología. Y todas las funciones definidas por puntos pueden provenir de datos

a. dentro del archivo de entrada
b. de otro archivo de datos por columnas
c. de archivos de mallas (`.msh` o `.vtk`)
d. de vectores de FeenoX (posiblemente modificados en tiempo de ejecución)

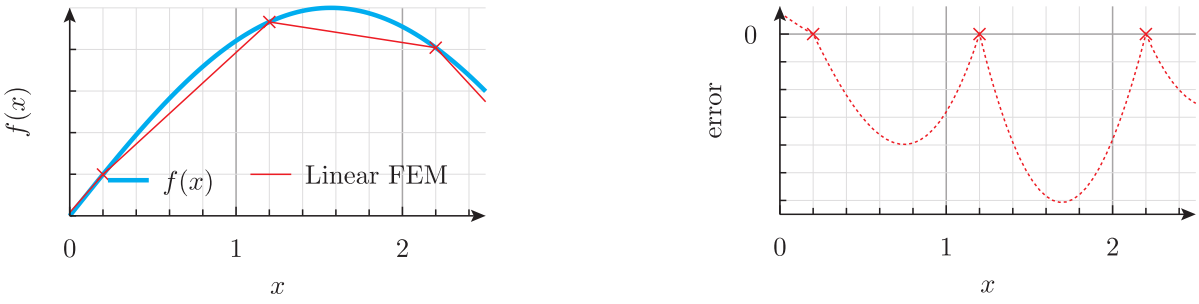
De hecho, aunque no provengan estrictamente de vectores (podrían hacerlo con la palabra clave `FUNCTION VECTOR`), FeenoX provee acceso a los vectores que contienen tanto los valores independientes (puntos de definición) como los valores dependientes (valores que toma la función). En la página 202 ilustramos este acceso.

Las funciones definidas por puntos que dependen de un único argumento tienen siempre una topología implícita. Para estos casos, FeenoX utiliza el framework de interpolación unidimensional `gsl_interp` de la GNU Scientific Library. En principio, en este tipo de funciones el nombre de los argumentos no es importante. Pero el siguiente ejemplo ilustra que no es lo mismo definir `f(x)` que `f(y)` ya que de otra manera la instrucción `PRINT_FUNCTION` daría idénticamente `sin(y)`, con el valor que tuviera la variable `y` al momento de ejecutar la instrucción `PRINT_FUNCTION` en lugar del perfil `sin(x)` que es lo que se busca:

```
FUNCTION f(x) DATA {
-0.8 sin(-0.8)
0.2 sin(0.2)
1.2 sin(1.2)
2.2 sin(2.2)
3.2 sin(3.2)
```

```
}

f_a = vecmin(vec_f_x)
f_b = vecmax(vec_f_x)
PRINT_FUNCTION f f(x)-sin(x) MIN f_a MAX f_b NSTEPS 100
```

Este pequeño archivo de entrada—que además muestra que una función definida por puntos puede usar expresiones algebraicas—fue usado para generar la figura 4.4.



(a) (b)

Figura 4.4.: Dos figuras para ilustrar que el error cometido en una aproximación lineal de elementos finitos es mayor lejos de los nodos fabricada a partir de datos numéricos generados por FeenoX.

Como mencionamos, las funciones definidas por puntos de varias dimensiones pueden tener o no una topología asociada. Si no la tienen, la forma más simple de interpolar una función de k argumentos $f(\mathbf{x})$ con $\mathbf{x} \in \mathbb{R}^k$ de estas características es asignar al punto de evaluación $\mathbf{x} \in \mathbb{R}^k$ el valor f_i del punto de definición $\mathbf{x}_i$ más cercano a $\mathbf{x}$.

Observación. La determinación de cuál es el punto de definición $\mathbf{x}_i$ más cercano a $\mathbf{x}$ se realiza en orden $O(\log N)$ con un árbol k-dimensional[36] conteniendo todos los puntos de definición $\mathbf{x}_i \in \mathbb{R}^k$ para $i = 1, \ldots, N$. La implementación del k-d tree no es parte de FeenoX sino de una biblioteca externa libre y abierta tipo "header only".

Observación. La noción de "punto más cercano" involucra una métrica del espacio de definición $\mathbb{R}^k$. Si las k componentes tienen las mismas unidades, se puede emplear la distancia euclidiana usual. Pero por ejemplo si una componente es una temperatura y otra una presión, la métrica euclidiana depende de las unidades en la que se expresan las componentes por lo que deja de ser apropiada.

Una segunda forma de evaluar la función es con una interpolación tipo Shepard [54], original o modificada. La primera consiste en realizar una suma pesada con alguna potencia p de la distancia del punto de evaluación $\mathbf{x}$ a todos los N puntos de definición de la función

$$f(\mathbf{x}) = \frac{\sum_{i=1}^{N} w_i(\mathbf{x}) \cdot f_i}{\sum_{i=1}^{N} w_i(\mathbf{x})}$$

donde

[36] Del inglés *k-dimensional tree*.

$$w_i(\mathbf{x}) = \frac{1}{|\mathbf{x} - \mathbf{x}_i|^p}$$

La versión modificada consiste en sumar solamente las contribuciones correspondientes a los puntos de definición que se encuentren dentro de una hiper-bola de radio R alrededor del punto de evaluación $\mathbf{x} \in \mathbb{R}^k$.

Observación. La determinación de qué puntos $\mathbf{x}_i$ están dentro de la hiper-bola de centro $\mathbf{x}$ y de radio R también se realiza con un árbol k-d.

4.3.3. Funciones definidas por puntos con topología implícita

Si los puntos de definición están en una grilla multidimensional estructurada rectangularmente (no necesariamente con incrementos uniformes), entonces FeenoX puede detectar la topología implícita y realizar una interpolación local a partir de los vértices del hiper-cubo que contiene el punto de evaluación $\mathbf{x} \in \mathbb{R}^k$. Esta interpolación local es similar a la explicada a continuación para el caso de topología explícita mediante una generalización de las funciones de forma para los elementos producto-tensor de primer orden a una dimensión arbitraria k.

Por ejemplo, si tenemos el siguiente archivo con tres columnas

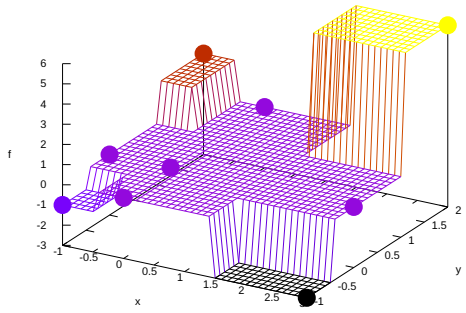
1. x_i
2. y_i
3. $f_i = f(x_i, y_i)$

```
-1 -1   -1
-1  0    0
-1 +2    2
 0 -1    0
 0  0    0
 0 +2    0
+3 -1   -3
+3  0    0
+3 +2   +6
```

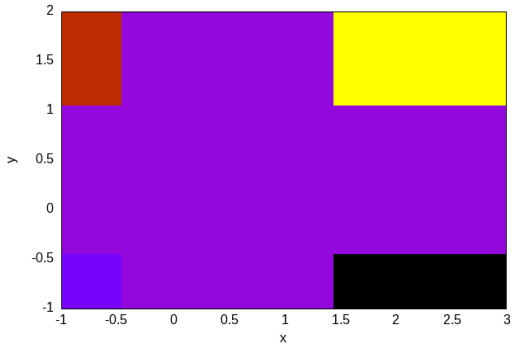
donde no hay una topología explícita pero sí una rectangular implícita, entonces podemos comparar las tres interpolaciones con el archivo de entrada

```
FUNCTION f(x,y) FILE hyperbolic-paraboloid.dat INTERPOLATION nearest
FUNCTION g(x,y) FILE hyperbolic-paraboloid.dat INTERPOLATION shepard
FUNCTION h(x,y) FILE hyperbolic-paraboloid.dat INTERPOLATION rectangular

PRINT_FUNCTION f g h MIN -1 -1 MAX 3 2 NSTEPS 40 30
```

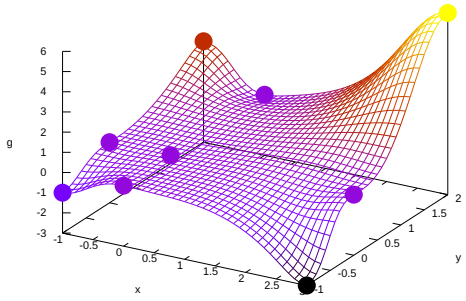
para obtener la figura 4.5.



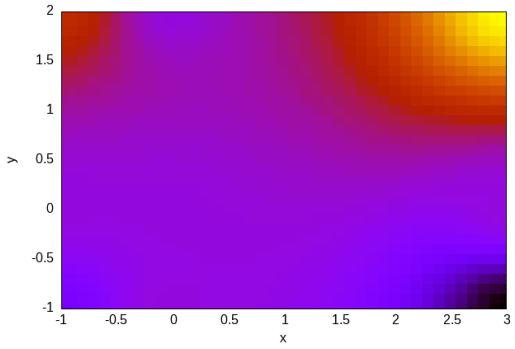
(a) $f(x, y)$ (`nearest`)



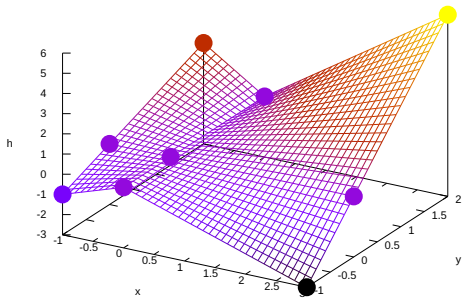
(b) $f(x, y)$ (`nearest`)



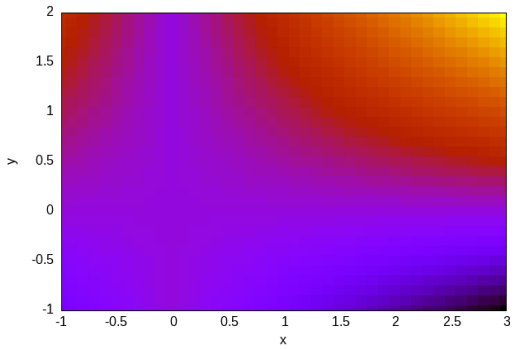
(c) $g(x, y)$ (`shepard`)



(d) $g(x, y)$ (`shepard`)



(e) $h(x, y)$ (`rectangular`)



(f) $h(x, y)$ (`rectangular`)

Figura 4.5.: Tres formas de interpolar funciones definidas por puntos a partir del mismo conjunto de datos con topología implícita.

4.3.4. Funciones definidas por puntos con topología explícita

Otra forma de definir y evaluar funciones definidas por puntos es cuando existe una topología explícita. Esto es, cuando los puntos de definición forman parte de una malla no estructurada con una conectividad conocida. En este caso, dada una función $f(\mathbf{x})$, el procedimiento para evaluarla en $\mathbf{x} \in \mathbb{R}^2$ o $\mathbf{x} \in \mathbb{R}^3$ es el siguiente:

1. Encontrar el elemento e_i que contiene al punto $\mathbf{x}$
2. Encontrar las coordenadas locales $\boldsymbol{\xi}$ del punto $\mathbf{x}$ en e_i
3. Evaluar las J funciones de forma $0 \leq h_j(\boldsymbol{\xi}) \leq 1$ del elemento e_i en el punto $\boldsymbol{\xi}$
4. Calcular $f(\mathbf{x})$ a partir de los J valores nodales de definición f_j como

$$f(\mathbf{x}) = \sum_{j=1}^{J} h_j(\boldsymbol{\xi}) \cdot f_j$$

La forma particular de implementar los puntos 1 y 2 (especialmente el 1) es crucial en términos de performance. FeenoX busca el elemento e_i con una combinación de un k-d tree para encontrar el nodo más cercano al punto $\mathbf{x}$ y una lista de elementos asociados a cada nodo. Una vez encontrado el elemento e_i, resuelve un sistema de ecuaciones de tamaño k (posiblemente no lineal) para encontrar las coordenadas locales $\boldsymbol{\xi} \in \mathbb{R}^k$.

De esta manera, si en lugar de tener los puntos de definición completamente estructurado de la página 202 tuviésemos la misma información pero en lugar de incluir el punto de definición $f(0,0) = 0$ tuviésemos $f(0.5, 0.5) = 0.25$ pero con la topología asociada (figura 4.6), entonces todavía podemos evaluar $f(\mathbf{x})$ en cualquier punto arbitrario $\mathbf{x} \in [0,1] \times [0,1]$ para obtener la figura 4.7:

```
READ_MESH 2d-interpolation-topology.msh DIM 2 READ_FUNCTION f
PRINT_FUNCTION f MIN -1 -1 MAX 3 2 NSTEPS 40 30
```

Esta funcionalidad permite realizar lo que se conoce como "mapeo de mallas no conformes" (ver sección 5.1). Es decir, utilizar una distribución de alguna propiedad espacial (digamos la temperatura) dada por valores nodales en una cierta malla (de un solver térmico) para evaluar propiedades de materiales (digamos la sección eficaz de fisión) en la malla de cálculo. En este caso en particular, los puntos $\mathbf{x}$ donde se requiere evaluar la función definida en la otra malla de definición corresponden a los puntos de Gauss de los elementos de la malla de cálculo. En el caso de estudio de la sección 5.1 (y en la sección B.3.2.2 del SDS) profundizamos este concepto.

Observación. Como mencionamos antes, es importante remarcar que para todas las funciones definidas por puntos, FeenoX utiliza un esquema de memoria en la cual los datos numéricos tanto de la ubicación de los puntos de definición $\mathbf{x}_j$ como de los valores f_j de definición están disponibles para lectura y/o escritura como vectores accesibles como expresiones en tiempo de ejecución. Esto quiere decir que esta herramienta puede leer la posición de los nodos de un archivo de malla fijo y los valores de definición de alguna otra fuente que provea un vector del tamaño adecuado, como por ejemplo un recurso de memoria compartida o un socket TCP [75], [82], [87]. Por ejemplo, podemos modificar el valor de definición $f(0.5, 0.5) = 0.25$ a $f(0.5, 0.5) = 40$ en tiempo de ejecución como

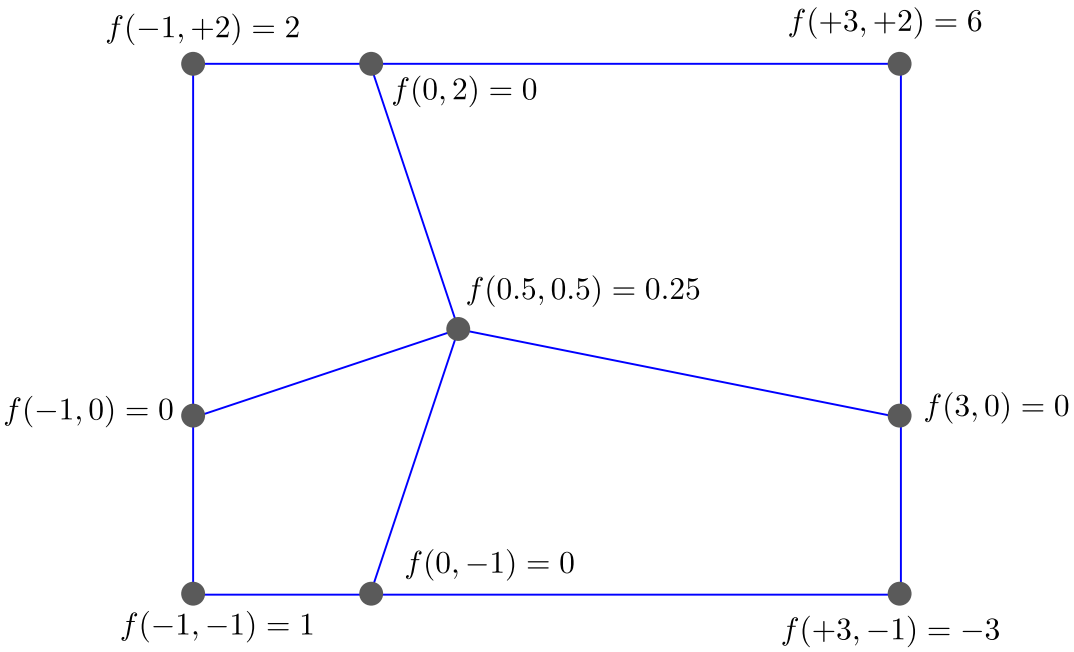
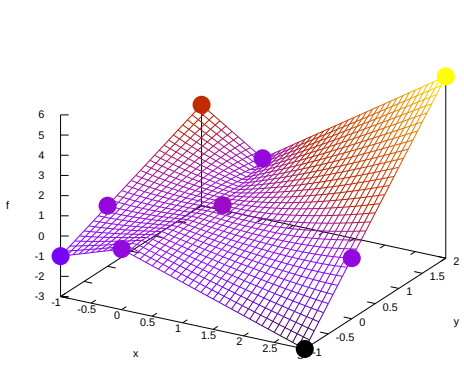
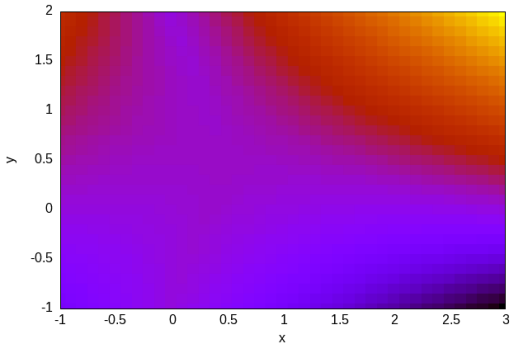


Figura 4.6.: Definición de una función $f(x,y)$ a partir de 9 datos discretos f_j y con una topología bi-dimensional explícita.



(a) $f(x,y)$ con topología

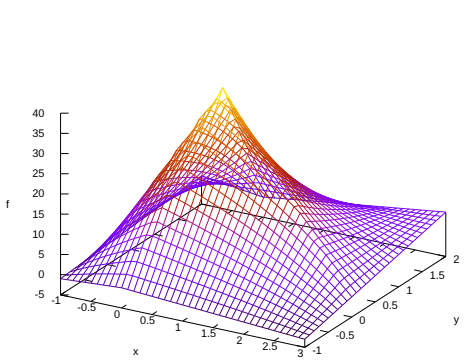


(b) $f(x,y)$ con topología

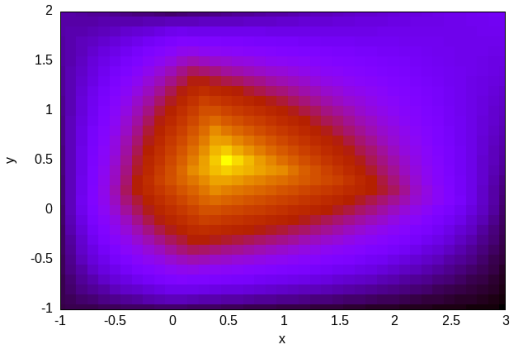
Figura 4.7.: Interpolación de una función definida por puntos con una topología explícita. La función interpolada coincide con la $h(x,y,)$ de la figura 4.5.

```
READ_MESH 2d-interpolation-topology.msh DIM 2 READ_FUNCTION f

# modificamos el valor de f(0.5,0.5) de 0.25 a 40
vec_f[5] = 40

PRINT_FUNCTION f MIN -1 -1 MAX 3 2 NSTEPS 40 30
```

para obtener la figura 4.8.



(a) $f(x, y)$ con topología y datos modificados



(b) $f(x, y)$ con topología y datos modificados

Figura 4.8.: Ilustración de la modificación en tiempo de ejecución de los datos de definición de una función definida con puntos (en este caso, con topología explícita).

4.4. Otros aspectos

Finalizamos este capítulo pasando revista a algunos aspectos de diversa importancia.

Observación. Hay varios otros aspectos de la implementación que, por cuestiones de límite de espacio y tiempo no explicamos. Por ejemplo, quedan fuera de la discusión que resta

- la forma de calcular campos secundarios con puntos de entrada para diferentes PDEs, incluyendo los algoritmos de extrapolación desde los puntos de Gauss a los nodos y promediado sobre nodos
- detalles de cómo se construyen los objetos algebraicos globales
- la forma de poner las diferentes condiciones de contorno
- la creación de las matrices jacobianas para problemas no lineales
- etc.

4.4.1. Licencia libre y abierta

Quisiera enfatizar en esta sección la importancia del software libre y abierto en general y del relacionado a cálculos de ingeniería en particular. Lo primero que hay que decir es que “software libre”

o "código abierto" no implica necesariamente el concepto de *gratuidad*. Afortunadamente, a diferencia de lo que sucede en inglés, en español los conceptos "libre" y "gratis" se denotan con palabras diferentes. Por lo tanto no debería haber tanta ambigüedad como la hay con el término *free software*. Sin embargo, más de una docena de años de experiencia me indican que esta diferencia no está suficientemente explicada, especialmente en el ambiente de la ingeniería mecánica donde también se confunden los conceptos de "elementos finitos" con "resolución de las ecuaciones de elasticidad con el método de elementos finitos".

Si bien estrictamente hablando los conceptos de "software libre" y "código abierto" provienen de etimologías e ideologías diferentes, prácticamente hablando hacen referencia a los mismos conceptos. Y dentro de todos los conceptos asociados, el más importante es el de libertad (freedom en inglés): la licencia de un cierto software es libre y abierta si le otorga al usuario las cuatro libertades fundamentales [85]:[37]

> 0. The freedom to run the program as you wish, for any purpose (freedom 0).
> 1. The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.
> 2. The freedom to redistribute copies so you can help others (freedom 2).
> 3. The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

Es decir, la importancia del software libre es que los usuarios, que justamente hacen uso del software para que una o más computadoras hagan una cierta tarea—en particular resolver una ecuación diferencial en derivadas parciales—tienen la posibilidad de ver exactamente cómo es que ese software resuelve dichas ecuaciones. E incluso tienen la posibilidad de modificar el código para que las resuelva como ellos quieren. Y acá viene el punto central de la idea. Por más que esos usuarios no tengan los conocimientos necesarios para modificar o incluso para entender el código fuente, tienen la *libertad* de contratar a alguien para que lo haga por ellos. Es por eso que la idea de libertad es central: en el concepto "free software" la palabra *free* se debe entender como en "free speech" y no como en "free beer" [88].

Observación. Las libertades dos y tres son esencialmente importantes en el ámbito académico.

Relacionado al movimiento de software libre, que tiene raíces en ideas éticas [57], viene el concepto de código abierto que se basa en consideraciones más bien prácticas: "given enough eyeballs all bugs are shallow" [46]. De hecho esta idea aplica también perfectamente al software de ingeniería: la calidad de un solver open source debería ser, objetivamente hablando, superior a cualquier otra herramienta privativa (en el sentido de que *priva* a los usuarios de las libertades básicas).

Una vez explicados las bases del software libre y del código abierto, quiero volver a aclarar por qué aquellos que deciden usar este tipo de programas basándose en consideraciones de precios están equivocados. El argumento es que pueden usar solvers "gratuitos" que "hacen lo mismo" que solvers

[37] Cita textual de "What is Free Software?" de la Free Software Foundation que nos hace acordar a la explicación de la ley cero de la termodinámica: *The reason they are numbered 0, 1, 2 and 3 is historical. Around 1990 there were three freedoms, numbered 1, 2 and 3. Then we realized that the freedom to run the program needed to be mentioned explicitly. It was clearly more basic than the other three, so it properly should precede them. Rather than renumber the others, we made it freedom 0.*

comerciales por los que hay que pagarle una licencia para poder usarlos al desarrollador o, peor aún, a un re-vendedor. Esto hace que este tipo de software, que elimina directamente la libertad básica cero sea clasificado como *privativo*, como ya explicamos en el párrafo anterior. Pero esto no quiere decir que

1. haya que aceptar cualquier tipo de software libre
2. el software libre no pueda ser "comercial"

Observación. Como explica técnicamente muy bien pero comunicacionalmente muy mal la Free Software Foundation, el software libre puede tener un modelo de negocios por detrás y ser efectivamente comercial. No hay ningún conflicto ético entre intentar obtener rédito económico escribiendo software y la filosofía de libertad que subyace bajo los ideales del software libre. Luego, es incorrecto usar la palabra "comercial" como antónimo de "software libre".

Observación. El gerente de ingeniería de una compañía americana que fabrica aleaciones de aluminio especiales con coeficiente de expansión negativo se contactó conmigo porque FeenoX le parecía una herramienta muy interesante para su empresa. Hasta ese momento, FeenoX solamente podía realizar cálculos con un único coeficiente de expansión térmica isotrópico. Pero esta compañía necesitaba realizar cálculos con coeficientes de expansión ortotrópicos. Luego de una propuesta de consultoría, agregué la funcionalidad requerida a cambio de un cierto pago a satisfacción del cliente, con la condición de que el código resultante fuese incorporado a la base de FeenoX con licencia GPLv3+. En forma similar, el autor de la tesis de doctorado [87] me obsequió una botella de cachaça a cambio de haber distribuido el software milonga (la versión anterior de FeenoX) bajo una licencia abierta. Por lo tanto, de primera mano puedo afirmar que el software libre no es incompatible con la idea de software "comercial".

Observación. Si un programa viene con su código fuente pero el desarrollador original pide que los usuarios firmen un NDA para poder ejecutarlo o impide que personas de cierta nacionalidad puedan usarlo, entonces puede llegar a ser de código abierto pero no es software libre ya que limita la libertad número cero.

Observación. Si un programa de cálculo viene con su código fuente pero éste está escrito en un dialecto de Fortran 77 ininteligible para alguien que haya nacido después de 1980, entonces no es software libre porque limita la libertad número uno.

Para redondear la idea, quisiera citar—una vez más—a Lucio Séneca, que ya nos explicaba la relación entre gratuidad y libertad hace dos mil años durante la Roma Imperial en una carta a su discípulo Lucilio:

> "A veces pensamos que sólo se compran las cosas por las que pagamos dinero y llamamos gratuitas a aquellas por las que terminamos sacrificando nuestras personas. Los productos que no estaríamos dispuestos a comprar si a cambio de ellos tuviéramos que entregar nuestra casa o un campo apacible o productivo, estamos muy resueltos a conseguirlos a costa de llenarnos de inquietudes, de peligros, de perder el honor, de perder nuestra libertad y nuestro tiempo; … actuemos, pues, en todos nuestros proyectos y asuntos igual que solemos hacerlo siempre que acudimos a un negocio: consideremos a qué precio se ofrece el objeto que deseamos. Con frecuencia tiene el máximo costo aquel por el que no se paga ningún precio. Podría mostrarte muchos regalos cuya adquisición y aceptación nos ha arrebatado la libertad. Seríamos dueños

> de nosotros si ellos no fueran nuestros."

Hay mucho (más de lo que debería) software de cálculo distribuido bajo licencias compatibles con el software libre pero cuya calidad deja mucho que desear y que, sin embargo, es usado por ingenieros con el equivocado concepto de gratuidad. Dejo entonces dos frases populares Argentinas para que cada uno de los amables lectores elija de qué lado se acuesta (con respecto al software libre):

> i. A caballo regalado no se le miran los dientes.
> ii. Cuando la limosna es grande, hasta el santo desconfía.

Observación. El software FeenoX se distribuye bajo licencia GNU General Public License versión tres o, a elección del usuario, cualquier versión posterior que esté disponible al momento de recibir el software. Esto implica que además de las cuatro libertades, el usuario que recibe una copia del software tiene una restricción: no puede re-distribuir el software si modifica la licencia. Todas las re-distribuciones, tanto con o sin modificaciones, deben ser realizadas bajo la misma licencia con la que fue recibida el software. Este concepto, denominado *copyleft* (que es otro juego de palabras como lo son Unix, Bash, GNU, less, etc.) hace que el dueño del *copyright* (en este caso este que escribe) justamente lo use para evitar que alguien que no lo tenga pueda transformar FeenoX en software privativo.

Observación. Siguiendo la recomendación de la organización GNU, al ejecutar `feenox` con la opción `-v` (o `--version`) no solamente se reporta la versión sino también el mensaje de derechos de copia y un resumen de la licencia:

```
$ feenox -v
FeenoX v1.0.8-g731ca5d
a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Copyright © 2009--2024 Seamplex, https://seamplex.com/feenox
GNU General Public License v3+, https://www.gnu.org/licenses/gpl.html.
FeenoX is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
$
```

4.4.2. Filosofía Unix

Por diseño, FeenoX ha sido escrito desde cero teniendo encuentra las ideas de la filosofía de programación Unix resumida en las 17 reglas discutidas el libro "The Art of Unix Programming"[38] [47] que explicamos brevemente en el Apéndice C. Los autores originales de Unix explican sus ideas en la referencia [50]. En particular, son de especial aplicación a FeenoX las reglas de

- composición (sección C.3)
- separación (sección C.4)
- simplicidad (sección C.5)
- parsimonia (sección C.6)

[38] Este título es un juego de palabras donde Eric Raymond le responde a la obra clásica (e inconclusa al momento de la escritura de esta tesis) de Donald Knuth "The Art of Computer Programming" [29].

- transparencia (sección C.7)
- mínima sorpresa (sección C.10)
- silencio (sección C.11)
- economía (sección C.13)
- generación (sección C.14)
- diversidad (sección C.16)

En particular, se hace especial énfasis en que el problema a resolver esté completamente definido en un archivo de texto tipo ASCII que pueda ser seguido con un sistema de control de versiones tipo Git. Las mallas, que no son amenas a Git, *no* son parte del archivo de entrada de FeenoX sino que son referenciadas a través de una ruta a un archivo separado. La idea es que la malla sea generada a partir de otro archivo ameno a Git, como por ejemplo los archivos de entrada de Gmsh o líneas de código en alguno de los lenguajes para los cuales Gmsh provee una API: Python, Julia, C o C++.

Otro ejemplo de ideas de Unix implementadas en FeenoX es la posibilidad de realizar estudios paramétricos leyendo parámetros por la línea de comandos, como explicamos en la sección 4.4.3 y que utilizamos extensivamente en el capítulo 5. Esto permite que los parámetros a evaluar puedan ser generados por scripts de Bash (que es lo que mayormente usamos en esta tesis) pero también en Python (ver sección 5.8.3).

La decisión de utilizar bibliotecas numéricas libres, abiertas y bien establecidas también—de alguna manera—responde a una de las ideas de la filosofía Unix: do not repeat yourself! No tiene ningún sentido ponerse a programar los métodos numéricos necesarios para resolver las ecuaciones algebraicas discretizadas desarrolladas en el capítulo 3. No sólo el trabajo ya está hecho y disponible en forma libre y abierta sino que es muy poco probable que el código propio sea más eficiente que el código de PETSc y SLEPc que involucra varios años-hombre de matemáticos y programadores profesionales. Más aún, si algún investigador (que tal vez es uno de estos mismos matemáticos o programadores) descubre algún método o algoritmo más eficiente, una actualización de la biblioteca proveería al solver neutrónico con estos nuevos métodos incrementando su performance casi automáticamente.

Observación. El autor del precondicionador GAMG de PETSc implementó en la versión 3.20 un nuevo esquema de *coarsening* que, para algunos problemas con ciertas opciones de optimización en el compilador, es más rápido que en la versión 3.19.

Observación. Justamente, las versiones 3.20 y 3.21 de PETSc incluyen contribuciones (corrección de bugs y tests de nuevos features) por parte autor de esta tesis (figura 4.9).

Observación. Al ejecutar `feenox` sin ninguna opción ni archivo de entrada, éste reportará en la terminal su versión—que incluye el hash del último commit del repositorio Git usado para compilar el ejecutable:

```
$ feenox
FeenoX v0.3.264-ge4ee9c5
a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool

usage: feenox [options] inputfile [replacement arguments] [petsc options]

  -h, --help         display options and detailed explanations of commmand-line usage
  -v, --version      display brief version information and exit
```

```
We recommend upgrading to PETSc 3.20.0 soon.
As always, please report problems to  petsc-maint at mcs.anl.gov
and ask questions at petsc-users at mcs.anl.gov

This release includes contributions from

David Ham Iglesia Dolci  Jack Betteridge Koki Sagiyama
Marcin Rogowski Nathan Collier Aidan Hamilton Albert Cowie
Barry Smith Blaise Bourdin Connor Ward David Andrs David Wells
Duncan Campbell Hansol Suh Hong Zhang Jacob Faibussowitsch
James Wright Jed Brown [redacted] Joseph Pusztay Jose Roman
Junchao Zhang Lisandro Dalcin Mark Adams Matthew Knepley
Mr. Hong Zhang Pablo Brubeck Pierre Jolivet Pieter Ghysels
Richard Tran Mills Satish Balay Sebastian Grimberg
Stefano Zampini Suyash Tandon Toby Isaac YANG Zongze Zach Atkins
```

(a) PETSc 3.20—29 de septiembre de 2023

```
We are pleased to announce the release of PETSc version 3.21.0.
We recommend upgrading to PETSc 3.21.0 soon. A reminder that
releases are at the end of March and September each year.

This release includes contributions from

Albert Cowie Alex Lindsay Barry Smith Blanca Mellado Pinto
David Andrs David Kamensky David Wells Fabien Evard
Fande Kong Hansol Suh Hong Zhang Ilya Fursa James Wright
Jed Brown Jeongu Kim Jeremy L Thompson Jose Roman
Junchao Zhang Koki Sagiyama Lars Bilke Lisandro Dalcin
Mark Adams Martin Diehl Massimiliano Leoni Matthew Knepley
Matt McGurn Mr. Hong Zhang  Nils Friess Pablo Brubeck
Pierre Jolivet René Chenard Rezgar Shakeri Richard Tran Mills
Satish Balay Sebastian Grimberg  Stefano Zampini Stephan Köhler
Toby Isaac YANG Zongze Zach Atkins
```

(b) PETSc 3.21—29 de marzo de 2024

Figura 4.9.: Anuncio de lanzamiento de PETSc con la lista de personas que han contribuido a la base del código, incluyendo al autor de esta tesis.

```
  -V, --versions     display detailed version information
  --pdes             list the types of PROBLEMs that FeenoX can solve, one per line
  --elements_info    output a document with information about the supported element types

Run with --help for further explanations.
$
```

Si se incluye la opción `-V` o `--versions` entonces se muestra más información sobre el ejecutable propiamente dicho y sus dependencias:

```
[redacted]chalmers:~$ feenox -V
FeenoX v0.3.267-g07ffa9e
a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Last commit date   : Thu Oct 26 12:47:53 2023 -0300
Build date         : Mon Oct 30 09:27:21 2023 -0300
Build architecture : linux-gnu x86_64
Compiler version   : gcc (Debian 12.2.0-14) 12.2.0
Compiler expansion : gcc -Wl,-z,relro -I/usr/include/x86_64-linux-gnu/mpich -L/usr/lib/x86_64- ↩
    linux-gnu -lmpich
Compiler flags     : -O3 -flto=auto -no-pie
Builder            : [redacted]tom
GSL version        : 2.7.1
SUNDIALS version   : N/A
PETSc version      : Petsc Release Version 3.20.0, Sep 28, 2023
PETSc arch         : double-int32-release
PETSc options      : --download-eigen --download-hdf5 --download-hypre --download-metis -- ↩
    download-mumps --download-parmetis --download-scalapack --download-slepc --with-64-bit- ↩
    indices=no --with-debugging=no --with-precision=double --with-scalar-type=real COPTFLAGS=- ↩
    O3 CXXOPTFLAGS=-O3 FOPTFLAGS=-O3
SLEPc version      : SLEPc Development GIT revision: v3.19.2-150-g8cd5b338c  GIT Date:  ↩
    2023-09-28 12:25:19 +0000
```

4.4.3. Simulación programática

Personalmente no me gusta el término "simulación",[39] pero el concepto de "simulación programática" es lo que se utiliza en la industria para indicar la posibilidad de realizar cálculos de ingeniería sin necesidad de una interfaz gráfica. En el mundo de software de ingeniería, esto involucra que los solvers provean

a. una interfaz en lenguaje de alto nivel, o
b. una entrada completamente definida en un archivo ASCII

En la mayoría de los programas industriales el camino para proveer "simulación programática" es agregar abstracciones e interfaces a software existente, muchas veces diseñados e implementados hace varias décadas. En FeenoX, esta idea está embebida en el diseño y se provee la "simulación programática" de forma nativa. De hecho en el año 2018 se ha desarrollado un proyecto industrial con la versión 2 del código en el cual un fabricante de implantes de cadera personalizados necesitaba incluir un paso de cálculo mecánico en su workflow automatizado sin intervención manual para definir el problema. La base de diseño del solver fue perfectamente adecuada para poder implementar dicho proyecto con una extrema satisfacción del cliente, acostumbrado a usar programas de cálculo tipo "point and click".

El hecho de diseñar el software comenzando por la idea de simulación programática en lugar de una interfaz gráfica hace que desarrollar una o más interfaces gráficas sea mucho más sencillo que el camino inverso tomado por las compañías de software de cálculo que se han dado cuenta de las ventajas de la "simulación programática". Más aún, dado que FeenoX está diseñado para correr en la nube (sección 4.4.6), es posible entonces desarrollar interfaces web en forma mucho más natural que si no se hubiesen tenido en cuenta todas estas consideraciones. La interfaz, web o desktop, tiene que hacer lo que haría un script programático (tal vez creando la malla y el archivo de entrada `.fee`) pero en forma interactiva.

Observación. La plataforma CAEplex lanzada en 2017 provee una interfaz web para una versión anterior de FeenoX que permite resolver problemas termo-mecánicos y modales en la nube directamente desde el navegador. Está 100% integrada en la plataforma CAD Onshape, tiene varios miles de usuarios registrados y al momento de la escritura de esta tesis contiene más de diez mil casos resueltos por los usuarios.

Observación. En la lista de trabajos futuros se incluye el desarrollo de interfaces para poder realizar definiciones y ejecutar instrucciones de FeenoX desde lenguajes de scripting, tal como hace Gmsh para Python y Julia.

4.4.4. Performance

Lo primero que hay que decir es que hay mucho "room for improvement" como dicen en inglés, especialmente en temas de performance. En general hay un balance entre requerimientos de CPU y memoria, y cada caso debe ser tratado en forma particular. Por ejemplo, si se almacenan todas las matrices elementales B_{gi} en cada uno de los puntos de Gauss entonces la recuperación de campos

[39] Buscar online el blog post "Say modeling not simulation" de 2017 para una explicación.

secundarios es más rápida a costa de incrementar (tal vez significativamente) la cantidad de memoria RAM necesaria.

Una de las premisas básicas de la optimización de código computacional es medir antes de intentar optimizar. Usando la biblioteca `benchmark` de Google[40] es posible realizar mediciones de secciones del código de FeenoX para comparar diferentes propuestas de optimización como trabajos a futuro:

- efecto de la posibilidad de que el compilar haga *inlining* de funciones usando *link-time optimizations*
- eficiencia de acceso a memoria para reducir la cantidad de *cache misses*
- re-diseñar estructuras de datos siguiendo los paradigmas del diseño orientado a datos (*data-oriented design*)
- estudiar pros y contras de usar el framework DMPlex [32] de PETSc para manejar la topología de las mallas no estructuradas

Otro de los balances que involucra a la performance de un código es la generalidad vs. la particularidad. Por ejemplo, la posibilidad de seleccionar en tiempo de ejecución cuál de todas las PDEs disponibles se quiere resolver involucra el esquema de apuntadores a función y de puntos de entrada discutidos en la sección 4.1.4. Esta generalidad hace que no se pueda hacer inlining de estas funciones ya que no se conocen en tiempo de compilación. Es por eso que tal vez se puede mejorar la eficiencia del código si la selección del tipo de PDE se pueda hacer en tiempo de compilación con macros apropiados que puedan optimizar para

- velocidad de ejecución
- memoria
- etc.

por ejemplo generando diferentes ejecutables de FeenoX para particularizaciones de

- el tipo de problema (para evitar apuntadores a función)
- la dimensión del problema (para poder usar arreglos de tamaño fijo en lugar de recurrir a reserva dinámica de memoria)
- mallas con todos los tipos de elementos iguales (para evitar tener que pedir memoria dinámica elemento por elemento)
- tamaño de variables de coma flotante (simple o doble precisión para optimizar memoria)
- etc.

Hay algunos estudios que muestran que para problemas de elasticidad lineal, FeenoX es tan o más rápido que otros programas de código abierto similares, incluso con todo el "room for improvement" de la versión actual.

Es preciso mencionar también que el tema de performance definido como "tiempo de ejecución requerido para resolver un cierto problema" es, por lo menos, ambiguo y difícil de cuantificar. Por ejemplo, consideremos el sistema de templates de C++. Estrictamente hablando, es un lenguaje Turing completo en sí mismo. Por lo tanto, es posible—al menos en teoría—escribir un solver para un problema particular (con la malla y condiciones de contorno embebidos en los templates) implementado 100% como templates de C++. En este caso inverosímil, el tiempo de ejecución sería cero ya que toda la complejidad numérica estaría puesta en la compilación y no en la ejecución.

[40] Ver repositorio https://github.com/seamplex/feenox-benchmark.

Si bien este experimento pensado es extremo, hay algunos puntos a tener en cuenta en casos reales. Consideremos el caso de solver algebraicos tipo multi-grid. El tiempo de CPU necesario para resolver un sistema de ecuaciones algebraicas depende fuertemente de la calidad de la malla. Entonces cabe preguntarse: ¿vale la pena "gastar" tiempo de CPU optimizando la calidad de la malla para que el solver converja más rápido? La respuesta va a depender de varias cuestiones, en particular si la misma malla va a ser usada una sola vez o hay varios problemas con diferentes condiciones de contorno que usan la misma malla. Una re-formulación de la conocida conclusión de que no existe el "one size fits all".

Por sobre este guiso de consideraciones, tenemos la salsa de la regla de economía de Unix. ¿Vale la pena emplear una cantidad x de horas de ingeniería para obtener un beneficio y en términos de recursos computacionales? Pregunta mucho más difícil de responder pero mucho más valiosa y apropiada. En la sección B.2 ilustramos brevemente esta idea. ¿Cuál es la combinación de herramientas que minimiza el costo total de resolver un laberinto arbitrario (figura B.7) en términos de tiempo de ingeniería más computación?

En relación directa a este concepto de economía de horas de ingeniería por horas de CPU está una de las ideas básicas del diseño de FeenoX. Estrictamente hablando es la regla del silencio de Unix pero fue una de las primeras lecciones aprendidas por este que escribe al trabajar en la industria nuclear con códigos escritos en la década de 1970. Como ya discutimos en el capítulo 1, en esos años cada hora de CPU era mucho más cara que cada hora del ingeniero a cargo de un cálculo. Es por eso que la regla de diseño de esos códigos de cálculo era "escribir en la salida todo lo que se calcula" ya que de necesitar un resultado calculado que no formara parte de la salida obligaría al ingeniero a volver a ejecutar el costoso cálculo. Hoy en día la lógica es completamente opuesta y, en general, es mucho más conveniente volver a realizar un cálculo que tener que buscar agujas en pajares ASCII de varios megabytes de tamaño. Es por eso que en FeenoX la salida está 100% definida en el archivo de entrada. Y de no haber ninguna instrucción tipo `PRINT` o `WRITE_RESULTS`, no habrá ninguna salida para el ingeniero.

Observación. Debido a que "todo es una expresión", FeenoX puede saber luego de parsear el archivo de entrada cuáles de los resultados obtenidos son usados en alguna salida. De esta manera puede tomar decisiones sobre si necesita calcular ciertos parámetros secundarios o no. Por ejemplo, si la función `Jx1` no aparece en ninguna expresión (incluyendo instrucciones de salida directa tales como `PRINT` o `WRITE_MESH` pero también instrucciones sin salidas directas como `INTEGRATE` o `FIND_EXTREMA`) entonces no llamará a las funciones relacionadas al cálculo de corrientes en difusión. Lo mismo sucede para los flujos de calor en el problema térmico y para las tensiones y deformaciones en elasticidad. En este caso la diferencia es aún más importante porque se distinguen los seis componentes del tensor simétrico de Cauchy y las tres tensiones principales σ_1, σ_2 y σ_3 que son los autovalores de dicho tensor. Si el usuario necesita las tensiones principales, `sigma1`, `sigma2` o `sigma3` aparecerán en al menos una expresión del archivo de entrada. Si no aparecen (y el parser de FeenoX lo sabe perfectamente) entonces no se desperdician ciclos de CPU calculando resultados que no se utilizan.

De todas maneras, no está de más estudiar detalladamente las formas de reducir el consumo de recursos computacionales para resolver un cierto problema de ingeniería, especialmente si el código va a ser usado masivamente en la nube. A la larga, esto repercutirá en menores costos y en menor consumo energético. Algunos puntos para continuar estudiando:

- El algoritmo de construcción de las matrices elementales para S_N es de lo más naïve y replica las ecuaciones algebraicas desarrolladas en el capítulo 3, lo que no suele ser una buena opción desde el punto de vista de análisis de algoritmos [29]. El tamaño de dichas matrices aumenta rápido con N y son, a la vez, esencialmente ralas.
- Aún cuando personalmente no comparta la idea de escribir nuevo código de cálculo en Fortran 77, debo reconocer que ésta tiene un punto interesante que debemos considerar. Dado que el modelo de memoria de Fortran 77 es muy limitado, el compilador puede hacer buenas optimizaciones automáticamente porque está seguro de que no habrá apuntadores apuntando a lugares inapropiados o de que no puede haber ciertas condiciones que, aunque poco probables, no permitan emplear algoritmos rápidos o utilizar eficientemente los registros del procesador. Esto en C no sucede automáticamente y es responsabilidad del programador emplear apropiadamente palabras clave reservadas como `const` y `restrict` para lograr el mismo nivel de optimización. Se deja también este análisis parte de los trabajos futuros.
- Para ciertos tipo de problemas matemáticos es conocido que el empleo de GPUs en lugar de CPUs puede reducir costos de ejecución ya que, bien empleadas, las GPUs proveen un factor de reducción de tiempos de cálculo mayor al factor de incremento de costos operacionales. Al utilizar PETSc, FeenoX puede correr los solvers algebraicos en GPU usando opciones en tiempo de ejecución. Si bien este concepto ha sido probado con éxito, se necesita más investigación para poder optimizar la ejecución en GPUs (o incluso en las nuevas APUs de reciente introducción en el mercado).
- Las bibliotecas de álgebra de bajo nivel tipo BLAS pueden ganar mucha eficiencia si son capaces de aprovechar las instrucciones tipo SIMD de los procesadores. Para ello es necesario configurar y compilar correctamente PETSc y sus dependencias según la arquitectura sobre la cual se va a ejecutar FeenoX (que no siempre es la arquitectura donde se compila). Hay varias implementaciones de bibliotecas tipo BLAS que PETSc puede usar (por ejemplo OpenBLAS, ATLAS, Netlib, MKL, etc.) cada una con diferentes opciones de compilación y características de optimización.

4.4.5. Escalabilidad

La idea de la escalabilidad de FeenoX viene de la posibilidad de resolver problemas arbitrariamente grandes mediante la paralelización con el estándar MPI [14], [84]. Tal como mostramos y discutimos en el capítulo 5, el hecho de que la ejecución en paralelo haga que el tiempo real (llamado también "tiempo de pared") necesario para obtener los resultados de resolver una ecuación en derivadas parciales es secundario con respecto al hecho de que la memoria por proceso MPI disminuye a medida que aumenta la cantidad de procesos para un problema de tamaño fijo.

Dicho esto, hay mucho trabajo por hacer en relación a la optimización de las ejecuciones en paralelo y en la interacción con bibliotecas de descomposición de dominios. Pero, como también mostramos en el capítulo 5, la funcionalidad básica de ejecución en paralelo se encuentra disponible en la versión actual de FeenoX.

Observación. Siguiendo las recomendaciones (tanto escritas como orales) de los desarrolladores de PETSc, no se considera en ningún momento la implementación de paralelización basada en

OpenMP[41]. Por un lado no hay evidencia de que este tipo de paralelización basada en threads provea una mejor performance que la paralelización de MPI basada en procesos. Por otro lado, el incremento de la cantidad de threads de OpenMP nunca va a redundar en una disminución de la memoria necesaria para resolver un problema de tamaño fijo ya que los threads son locales y no pueden ser separados en diferentes hosts con diferentes características de memoria RAM.

A modo de ilustración de las características de ejecución con MPI de FeenoX, consideremos el siguiente archivo de entrada (que es parte de los tests de FeenoX):

```
PRINTF_ALL "Hello MPI World!"
```

La instrucción `PRINTF_ALL` hace que todos los procesos escriban en la salida estándar los datos formateados con los especificadores de `printf` las variables indicadas, prefijando cada línea con la identificación del proceso y el nombre del host. Al ejecutar FeenoX con este archivo de entrada con `mpiexec` en un servidor de AWS apropiadamente configurado para que pueda conectarse a otro y repartir la cantidad de procesos MPI obtendríamos por ejemplo:

```
ubuntu@ip-172-31-44-208:~/mpi/hello$ mpiexec --verbose --oversubscribe --hostfile hosts -np 4 ./ ↩
    feenox hello_mpi.fee
[0/4 ip-172-31-44-208] Hello MPI World!
[1/4 ip-172-31-44-208] Hello MPI World!
[2/4 ip-172-31-34-195] Hello MPI World!
[3/4 ip-172-31-34-195] Hello MPI World!
ubuntu@ip-172-31-44-208:~/mpi/hello$
```

Esto es, el host `ip-172-31-44-208` crea dos procesos de `feenox` y a su vez le pide a `ip-172-31-34-195` que cree otros dos procesos. Estos podrán entonces resolver un problema en paralelo donde la carga de CPU y la memoria RAM se repartirán entre dos servidores diferentes.

Podemos utilizar el tutorial `t21` de Gmsh en el que se ilustra el concepto de DDM (descomposición de dominio o partición de la malla[42]) para mostrar otro aspecto de cómo funciona la paralelización por MPI en FeenoX. En efecto, consideremos la malla de la figura 4.10 que consiste en dos cuadrados adimensionales de tamaño 1×1 y supongamos que queremos integrar la constante 1 sobre la superficie para obtener como resultado el valor numérico 2:

```
READ_MESH t21.msh
INTEGRATE 1 RESULT two
PRINTF_ALL "%g" two
```

En este caso, la instrucción `INTEGRATE` se calcula en paralelo donde cada proceso calcula una integral local y antes de pasar a la siguiente instrucción, todos los procesos hacen una operación de reducción mediante la cual se suman todas las contribuciones y todos los procesos obtienen el valor global en la variable `two`:

```
$ mpiexec -n 2 feenox t21.fee
[0/2 tom] 2
[1/2 tom] 2
```

[41] No confundir con OpenMPI que es una de las varias implementaciones del estándar MPI y que sí es soportada por FeenoX.

[42] Del inglés *mesh partitioning*.

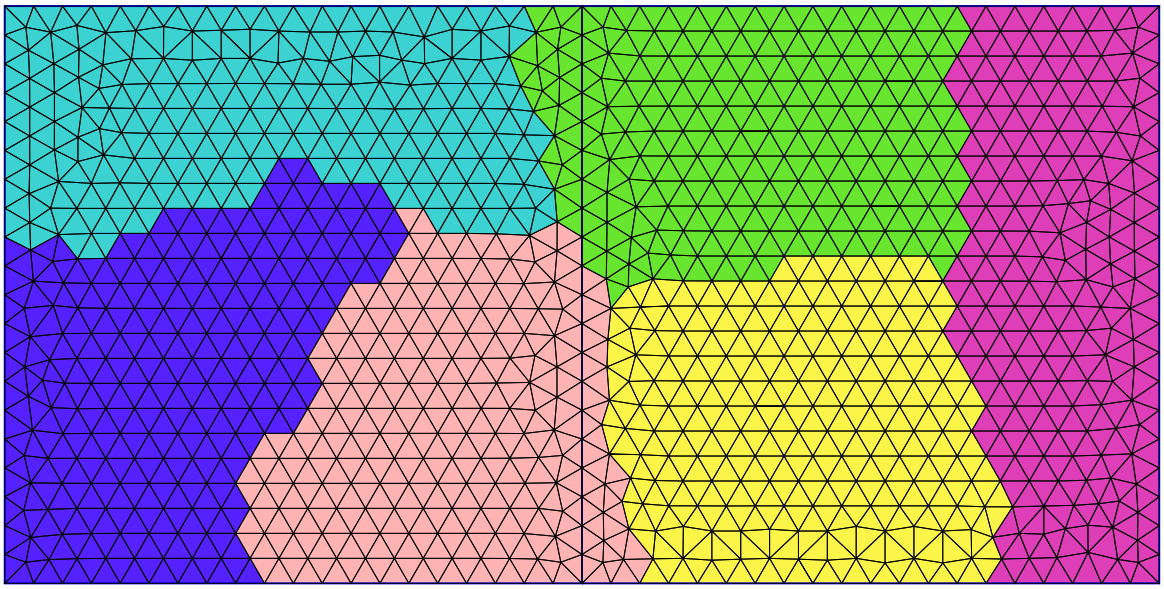


Figura 4.10.: Geometría tutorial `t21` de Gmsh: dos cuadrados mallados con triángulos y descompuestos en 6 particiones.

```
$ mpiexec -n 4 feenox t21.fee
[0/4 tom] 2
[1/4 tom] 2
[2/4 tom] 2
[3/4 tom] 2
$ mpiexec -n 6 feenox t21.fee
[0/6 tom] 2
[1/6 tom] 2
[2/6 tom] 2
[3/6 tom] 2
[4/6 tom] 2
[5/6 tom] 2
$
```

Para ver lo que efectivamente está pasando, modificamos temporalmente el código fuente de FeenoX para que cada proceso muestre la sumatoria parcial:

```
$ mpiexec -n 2 feenox t21.fee
[proceso 0] mi integral parcial es 0.996699
[proceso 1] mi integral parcial es 1.0033
[0/2 tom] 2
[1/2 tom] 2
$ mpiexec -n 3 feenox t21.fee
[proceso 0] mi integral parcial es 0.658438
[proceso 1] mi integral parcial es 0.672813
[proceso 2] mi integral parcial es 0.668749
[0/3 tom] 2
[1/3 tom] 2
[2/3 tom] 2
$ mpiexec -n 4 feenox t21.fee
[proceso 0] mi integral parcial es 0.505285
[proceso 1] mi integral parcial es 0.496811
```

```
[proceso 2] mi integral parcial es 0.500788
[proceso 3] mi integral parcial es 0.497116
[0/4 tom] 2
[1/4 tom] 2
[2/4 tom] 2
[3/4 tom] 2
$ mpiexec -n 5 feenox t21.fee
[proceso 0] mi integral parcial es 0.403677
[proceso 1] mi integral parcial es 0.401883
[proceso 2] mi integral parcial es 0.399116
[proceso 3] mi integral parcial es 0.400042
[proceso 4] mi integral parcial es 0.395281
[0/5 tom] 2
[1/5 tom] 2
[2/5 tom] 2
[3/5 tom] 2
[4/5 tom] 2
$ mpiexec -n 6 feenox t21.fee
[proceso 0] mi integral parcial es 0.327539
[proceso 1] mi integral parcial es 0.330899
[proceso 2] mi integral parcial es 0.338261
[proceso 3] mi integral parcial es 0.334552
[proceso 4] mi integral parcial es 0.332716
[proceso 5] mi integral parcial es 0.336033
[0/6 tom] 2
[1/6 tom] 2
[2/6 tom] 2
[3/6 tom] 2
[4/6 tom] 2
[5/6 tom] 2
$
```

Observación. En los casos con $M = 4$ y $M = 5$ procesos, la cantidad de particiones P no es múltiplo de la cantidad de procesos N. De todas maneras, FeenoX es capaz de distribuir la carga entre los N procesos requeridos aunque la eficiencia es menor que en los otros casos.

Al construir los objetos globales `K` y `M` o **b**, FeenoX utiliza un algoritmo similar para seleccionar qué procesos calculan las matrices elementales de cada elemento. Una vez que cada proceso ha terminado de barrer los elementos locales, se le pide a PETSc que ensamble la matriz global enviando información no local a través de mensajes MPI de forma tal de que cada proceso tenga filas contiguas de los objetos globales.

Observación. Cada fila de la matriz global `K` corresponde a un grado de libertad asociado a un nodo de la malla.

4.4.6. Ejecución en la nube

Observación. Esta sección podría ser una tesis académica completa (tal vez en el ámbito de tecnologías de informática y comunicaciones) o un informe técnico industrial en sí misma.

FeenoX es una herramienta computacional que ha sido diseñada, haciendo una analogía con el diseño web, como *cloud-first* (también llamados *API-first* en la industria del software) y no solamente como *cloud-friendly* (ver sección B.1 para una discusión más detallada sobre la diferencia entre este conceptos). Lo segundo quiere decir que la herramienta pueda ser ejecutada en servidores remotos de una forma más o menos sencilla. Pero la primera idea implica conceptos y decisiones de diseño más profundas, que explicamos en esta sección.

Lo primero que hay que decir es que cuando nos referimos a la "nube", desde un punto de vista de computación de alta performance, estamos haciendo referencia a que en principio disponemos de infinitos recursos computacionales. Haciendo una analogía que termina muy rápido, comparar recursos computacionales *on premise* con la nube equivale a comparar la generación eléctrica mediante máquinas propias (¿un generador Diesel?) con la posibilidad de tomar energía de la red eléctrica. Esto es, para la mayoría de las aplicaciones, recurrir a recursos computacionales cloud debería ser la primera opción tanto desde el punto de vista técnico como económico.

Para poder usufructuar las ventajas que este tipo de hardware provee, es mandatorio que el software pueda ejecutarse no sólo en forma remota sino que también sea capaz de correr en forma distribuida. Entonces, hay que

- tener todos los hosts en una red particular
- configurar sistemas de nombres de dominios
- diseñar sistemas de archivos de red compartidos
- etc.

En lugar de tener que hacer todo este set-up en forma manual cada vez que se necesite realizar un cálculo de ingeniería, una implementación cloud completa implicaría el desarrollo de una serie de scripts encargados de lanzar y configurar las instancias necesarias para ejecutar dicha simulación. Estos scripts se suelen conocer como "thin clients", podrían simplemente encargarse de

1. lanzar y configurar instancias remotas
2. subir archivos de entrada a dichas instancias
3. ejecutar las herramientas computacionales (Gmsh, FeenoX, etc.)
4. bajar los resultados

Pero también podrían diseñarse clientes (y servidores) más complejos que incluyan llamadas a APIs tipo REST relacionadas a la simulación en sí. Por ejemplo, que manejen temas como

- autenticación
- facturación de horas de CPU
- estimación de los recursos que deben ser lanzados en función del tipo y tamaño del problema a resolver
- estudios paramétricos
- lazos de optimización
- simulaciones condicionalmente encadenadas
- etc.

Si bien esta tesis no abarca a estos clientes (que queda como trabajo a futuro), el diseño de FeenoX es tal que su desarrollo es perfectamente posible y eficiente. De hecho nos referimos a "clientes" en plural porque, tal como pide la regla de Unix de diversidad (sección C.16), no hay un único tipo

de cliente posible sino que hay muchas dependiendo del tipo de problema a resolver. Y como FeenoX justamente es lo suficientemente flexible como para resolver no solamente diferentes PDEs sino también diferentes clases de problema (acoplados, paramétricos, de optimización, etc.) en diferentes entornos (muchos cálculos pequeños, pocos grandes, uno inmenso, etc.) bajo diferentes condiciones (en la industria por una sola persona, en la academia por un equipo, como una plataforma pública *as a service*, etc.) entonces es de esperar que no haya un cliente que pueda manejar todas las combinaciones en forma óptima. Pero sí lo que se ha tenido en cuenta en el diseño del código computacional de cálculo es que, una vez más siguiendo la filosofía Unix planteada implícitamente durante la década de 1970 [50] pero que sigue siendo extremadamente importante en la década de 2020 a la luz de la arquitectura que "la nube" fue tomando, se debe separar el *back end* de las capas de abstracción necesarias para llegar a los distintos *front ends* (figura 4.11) necesarios para su aplicación (regla de separación, sección C.4) .

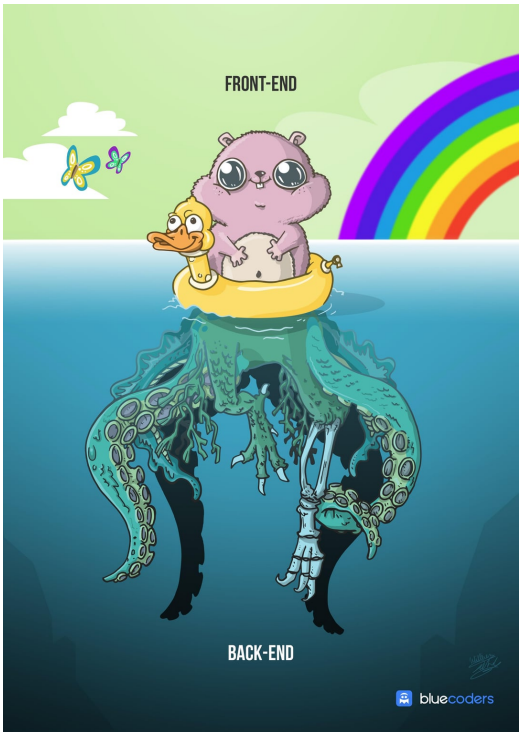


Figura 4.11.: Ilustración conceptual de la diferencia entre un *front end* y un *back end* ©bluecoders.

Observación. Este párrafo no es trivial. Conozco de primera mano los esfuerzos realizados por una de las empresas de software de cálculo más grandes del mundo (con facturación superior a los 2 mil millones de dólares en 2022 y comprada por 38 mil millones en 2024) que ha intentado agregar esta capa de abstracción a sus solvers existentes con una tasa de suceso tan baja que ha debido comprar una startup que estaba en mejores condiciones de hacerlo por más de cien millones de dólares. En una presentación del gerente encargado de la compra al comité de directores de la empresa (los famosos CxOs), una filmina[43] decía literalmente

[43] Por supuesto que "filmina" es un anacronismo deliberado. Fue una presentación hecha en un software privativo que no quisiera nombrar para no alentar a nadie a que lo use. Pero justamente, mi opinión personal es que el hecho de que la política de la compañía era usar este software privativo para hacer presentaciones es el que la ha impedido lograr desarrollar la capa de abstracción por la que han debido comprar una startup que hacía sus presentaciones con los macros Beamer de LaTeX.

> Desktop solvers design is opposed to API-first solvers.

¿Qué implica todo esto de API-first para un solver neutrónico? Que además de leer uno o más archivos de entrada (o instrucciones en un lenguaje de alto nivel como Python a través de una API) que definan el problema a resolver como suele suceder en la mayoría de los solvers de uso masivo, hay que tener en cuenta que durante la ejecución propiamente dicha se debe poder interactuar con distintas clases de entidades, como por ejemplo

- una interfaz de usuario web
- un sistema de logueo distribuido y compartido
- un reportero de estado bajo demanda por email, sms, whatsapp, telegram, etc.

Luego el back-end debe ser lo suficientemente flexible como para poder reportar su estado y sus eventos con un horizonte de observabilidad mucho mayor que el que se necesitaba en 1970 cuando se diseñaron muchas de las herramientas de uso nuclear (y no nuclear) industrial.

Especialmente importante es el manejo y reporte de errores. Mi experiencia profesional dice que los usuarios hacen gala de su denominación y usan el software de formas que no pueden ser tenidas en cuenta ni por un único desarrollador ni por un equipo de programadores profesionales. Algunas de estas formas tienen sentido y deben ser manejadas por el software. Pero la gran mayoría simplemente son entradas que no tienen sentido (y que justamente por eso no han sido tenidas en cuenta por los desarrolladores). Por lo tanto constantemente aparecen nuevos errores que deben ser reportados en forma humanamente entendible. Entonces tanto el back-end como la capa de abstracción deben ser, una vez más, lo suficientemente flexibles para manejar casos inesperados.

Para terminar, resaltamos que una de las varias maneras de hacer el *deployment* involucra utilizar la tecnología de contenedores. FeenoX es completamente Docker-friendly en el sentido de que el código puede ser

- obtenido desde el repositorio
- compilado (con las opciones particulares de optimización apropiadas para el hardware donde se va a terminar ejecutando el binario) y
- ejecutado

con líneas de comando estándar de Unix como mostramos en la sección 4.4.8.

4.4.7. Extensibilidad

Uno de los requerimientos del *Sofware Design Requirements* del apéndice B es que la herramienta computacional desarrollada pueda ser extensible. El esquema por el cual es posible agregar nuevas ecuaciones diferenciales en derivadas parciales a resolver con el método de elementos finitos con apuntadores a función ya ha sido explicado durante este capítulo. Esta es una característica distintiva que no es común en el mundo del software de elementos finitos, excepto en aquellas herramientas avanzadas como FEniCSx que permiten dar la forma débil de la ecuación a resolver en el archivo de entrada [2].

De todas maneras, si bien FeenoX tiene funcionalidades que no han sido discutidas en detalle en esta tesis (por ejemplo el hecho de poder resolver sistemas de ecuaciones algebraicas-diferenciales que mostramos brevemente en la sección 5.11), hay muchas otras características que serían deseables en una herramienta de este tipo que no están implementadas. Pero, en principio, la arquitectura del código permite que el desarrollo de temas como

- solución de PDEs discretizadas espacialmente con volúmenes finitos en lugar de elementos finitos
- esquemas espaciales de alto orden
- manejo de elementos estructurales (beam, truss, shell, plate) para resolver problemas de elasticidad
- análisis modales con amortiguación que pueden dar lugar a soluciones complejas
- neutrónica cinético-espacial

Observación. El Ing. Nuclear Ramiro Vignolo ha desarrollado

a. una prueba de concepto para resolver neutrónica a nivel de celda con el método de probabilidad de colisiones y
b. un solver de redes termohidráulicas 1D en estado estacionario

en la segunda versión del código, mostrando que ya era posible la extensibilidad en la arquitectura anterior aún cuando todavía no era éste uno de los puntos de la base de diseño.

En las conclusiones del capítulo 6 hacemos un listado extensivo de posibles características que podrían ser implementadas sin necesidad de realizar grandes refactorizaciones al código base.

Observación. Como ya hemos explicado en la sección 4.4.1, FeenoX se distribuye bajo licencia GPLv3 o posterior. Técnicamente hablando (en el sentido leguleyo), el "o posterior" es por extensibilidad (sección C.17).

4.4.8. Integración continua

Una de las tantas prácticas que se han puesto de moda en la última década en la industria del software que realmente agrega calidad al código resultante (al contrario que las otras modas como el sustantivo scrum y el adjetivo agile) es la llamada integración continua. Esta práctica involucra primero generar y luego automatizar muchos pasos de prueba del código antes de liberar públicamente las versiones.

Como ya mencionamos, el desarrollo de FeenoX se realiza en un repositorio Git alojado en Github (figura 4.12), pero que puede ser clonado y replicado libremente (siguiendo la licencia GPLv3+). Cada commit al repositorio tiene un hash asociado que, como también ya hemos mencionado, es reportado por el binario `feenox` tanto si se ejecuta sin argumentos como si se ejecuta con la opción `-v` (o `--version`). Más aún, la opción `-V` (o `--versions`) da no sólo el hash sino también la fecha y hora del último commit del repositorio utilizado para compilar el binario. De esta forma es posible vincular un ejecutable cualquiera encontrado "en la naturaleza"[44] con el estado instantáneo del código fuente a través del hash (ayudándose de la fecha reportada por `-V`).

[44] Del inglés *in the wild*.

Figura 4.12.: © Ladsgroup, CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0, via Wikimedia Commons

Observación. A lo largo de la historia del desarrollo de las tres versiones del código hemos utilizado los siguientes sistemas de control de versiones

- Subversion `svn`
- Bazaar `bzr`
- Mercurial `hg`

Al comienzo del desarrollo (2009) estuvimos a punto de emplear CVS, pero nos decidimos por `svn` por considerar que `cvs` era muy anticuado.[45]

Además del target `all` en el `Makefile` que compila el ejecutable, existe un target `check` que ejecuta una serie de scripts que corresponden al paso de testeo siguiendo el esquema de Autotools. Éste consiste en un conjunto de casos de prueba. Cada uno de ellos ejecuta `feenox` con diferentes archivos de entrada y compara la salida con resultados pre-definidos. Si la comparación no es exitosa (o si hay algún problema en tiempo de ejecución como por ejemplo una falta de segmentación) entonces el test se marca como fallado. Muchos de estos casos de prueba resuelven problemas cuyo resultado es conocido, sea porque tiene solución analítica o porque su solución numérica está bien establecida. Pero otros casos son arbitrarios y el resultado "de referencia" se utiliza para saber si alguna modificación posterior a la introducción de dicho test hace que el código arroje un resultado diferente. En el caso de que uno de estos casos—conocidos como "tests de regresión"[46]—falle, se debe

1. identificar el commit que hace que el test falle (el procedimiento usual es usar la facilidad de "bisección" de Git)
2. analizar si la falla del test es debido a un error en el commit o debido a que el resultado de referencia era incorrecto
3. modificar el código o el resultado de referencia
4. volver a ejecutar el target `check`

Empleando la característica "Actions" de Github, cuando cada uno de las ramas secundarias se une (¿mergea?) a la rama principal (main branch) se crea un servidor virtual básico con Ubuntu (porque no hay imágenes de Debian disponibles) y luego, en forma automática,

[45] El autor de esta tesis llegó a usar CVS en el siglo pasado.
[46] Del inglés *regression tests*.

1. se clona el repositorio

```
git clone https://github.com/seamplex/feenox
```

2. se instalan las dependencias necesarias desde Apt

```
sudo apt-get install -y libgsl-dev libsundials-dev petsc-dev slepc-dev gmsh
```

3. se hace el bootstrapping (figura 4.3), configuración y compilación del código

```
./autogen.sh && ./configure && make
```

4. se ejecuta el conjunto de tests y, en caso de al menos uno falle, se reporta el archivo de log

```
make check || (cat ./test-suite.log && exit 1)
```

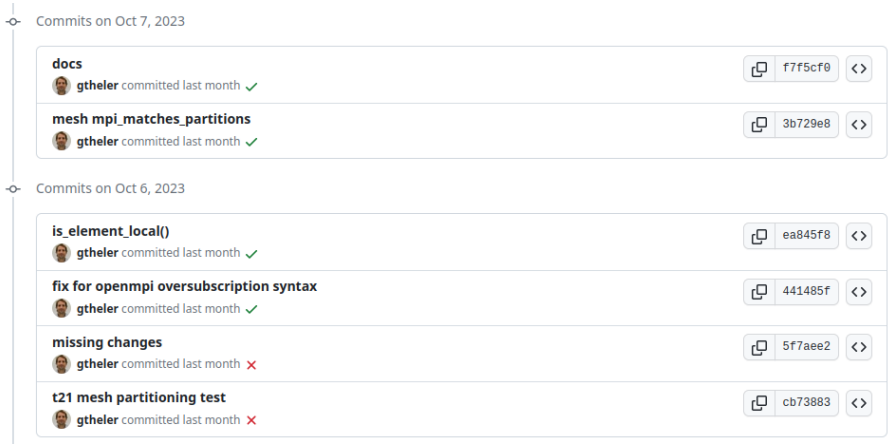


Figura 4.13.: Historial de commits en Github indicando tests pasados y fallados.

De esta manera, cualquier persona del mundo puede ver a través de la interfaz de Github los commits en los cuales al menos unos de los tests ha fallado (figura 4.13). Pero además, en caso de que algún commit en el branch `main` no pase los tests, la plataforma le envía un correo electrónico a los administradores del proyecto avisándole de esta situación para que se puedan tomar las decisiones apropiadas.

Si bien el comando `make check` ejecuta más de 350 casos, el código aún no está instrumentado para medir cuántas líneas son efectivamente "cubiertas" por los tests. Este trabajo de implementar lo que se conoce como medir el "code coverage" en la jerga de integración continua queda como trabajo a futuro (capítulo 6). De la misma manera, también queda como trabajo a futuro diseñar un conjunto de tests que corran bajo la herramienta de desarrollo `valgrind` para detectar sistemáticamente potenciales problemas con el manejo de memoria, incluyendo

- escrituras en direcciones de memoria no reservadas,
- des-referenciación de punteros inválidos, y/o
- pérdidas de memoria.[47]

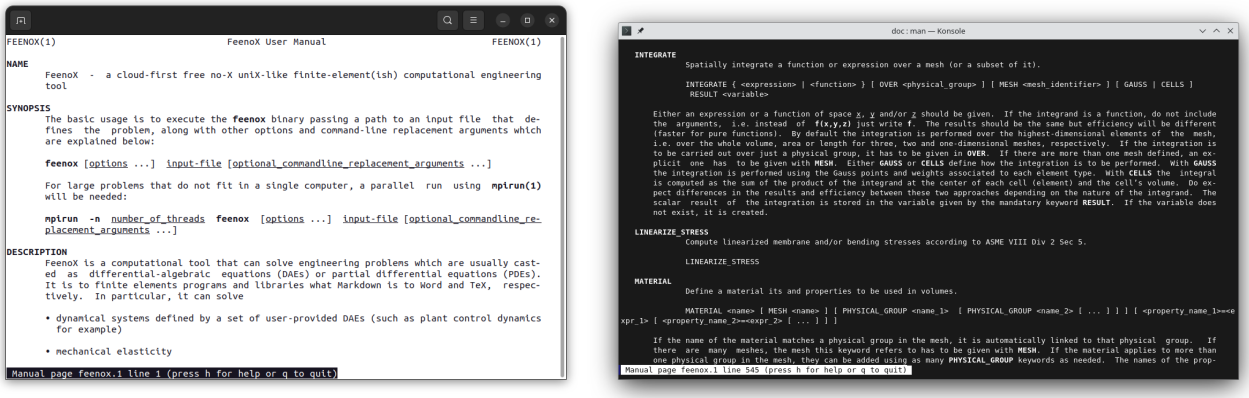
[47] Del inglés *memory leaks*

4.4.9. Documentación

El 100% de la documentación, incluyendo

- el sitio web seamplex.com/feenox
- el README del repositorio Git
- el manual descriptivo (en formato Texinfo)
- la referencia completa (en formato HTML y PDF)
- la página de manual de Unix (figura 4.14)
- el *Sofware Requirements Specification*
- el *Sofware Design Specification*
- los ejemplos
- esta misma tesis

provienen de texto escrito en Markdown. Luego mediante scripts específicamente diseñados, se crean los distintos documentos en los formatos apropiados con la herramienta Pandoc.



(a) Gnome Terminal

(b) Konsole

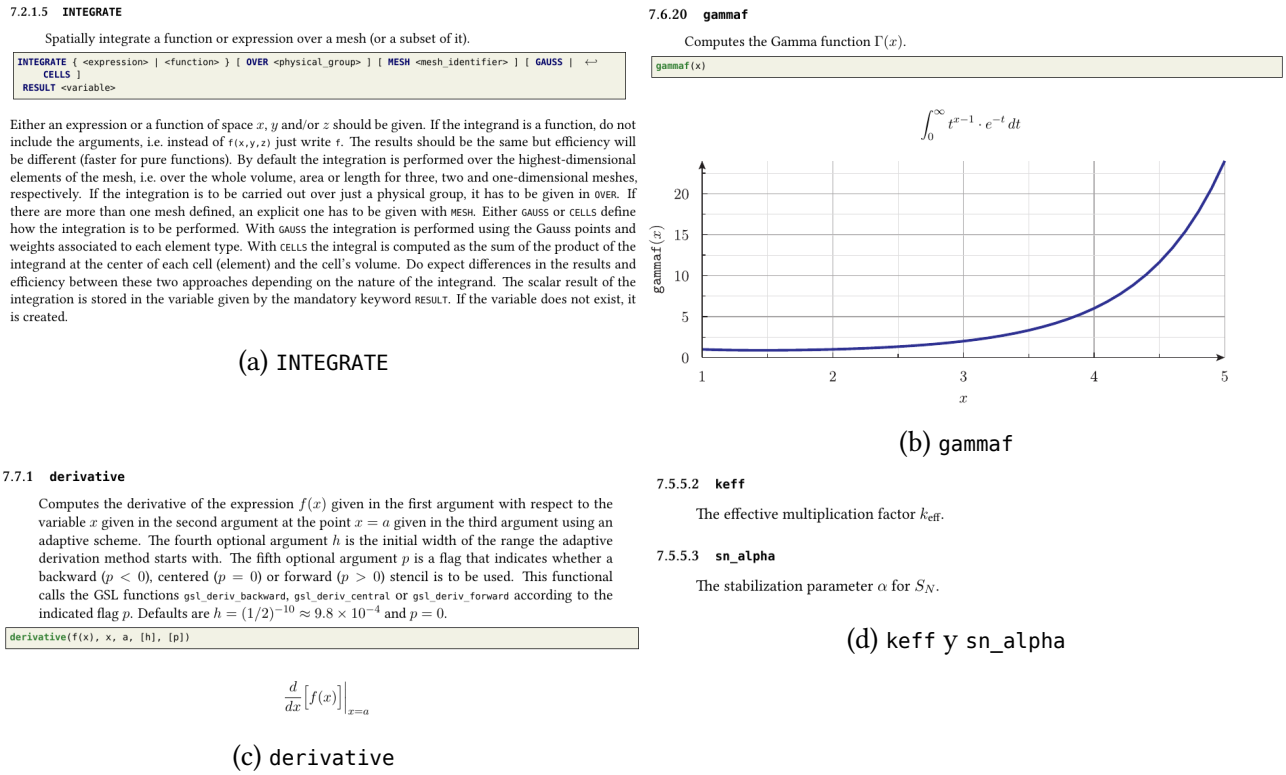
Figura 4.14.: La página de manual de FeenoX de Unix al ejecutar `man feenox`

Toda la documentación en Markdown (aún el fuente del sitio web, incluyendo ejemplos, tutoriales, etc.) forma parte del repositorio de FeenoX. El script que genera los archivos finales en HTML y en PDF inserta en el pie de cada página el hash y la fecha del último commit. De esta manera, si aparece un PDF en Scribd, uno puede saber cabalmente a qué versión de FeenoX se refiere.

El manual de referencia que indica los argumentos que toman las palabras clave, las variables especiales de cada PDE, las funciones internas, etc. provienen de comentarios especiales en el código fuente que comienzan con tres barras hacia adelante (en lugar de los comentarios regulares que usan dos barras). Estos comentarios incluyen meta-datos en un cierto formato que luego un script parsea y genera automáticamente texto en Markdown que luego es compilado al formato final. Por ejemplo,

- palabras clave

```
///kw_pde+INTEGRATE+usage { <expression> | <function> }
///kw_pde+INTEGRATE+detail Either an expression or a function of space $x$, $y$ and/or ↩
    $z$ should be given.
```



7.2.1.5 **INTEGRATE**

Spatially integrate a function or expression over a mesh (or a subset of it).

```
INTEGRATE { <expression> | <function> } [ OVER <physical_group> ] [ MESH <mesh_identifier> ] [ GAUSS |
    CELLS ]
RESULT <variable>
```

Either an expression or a function of space x, y and/or z should be given. If the integrand is a function, do not include the arguments, i.e. instead of `f(x,y,z)` just write `f`. The results should be the same but efficiency will be different (faster for pure functions). By default the integration is performed over the highest-dimensional elements of the mesh, i.e. over the whole volume, area or length for three, two and one-dimensional meshes, respectively. If the integration is to be carried out over just a physical group, it has to be given in `OVER`. If there are more than one mesh defined, an explicit one has to be given with `MESH`. Either `GAUSS` or `CELLS` define how the integration is to be performed. With `GAUSS` the integration is performed using the Gauss points and weights associated to each element type. With `CELLS` the integral is computed as the sum of the product of the integrand at the center of each cell (element) and the cell's volume. Do expect differences in the results and efficiency between these two approaches depending on the nature of the integrand. The scalar result of the integration is stored in the variable given by the mandatory keyword `RESULT`. If the variable does not exist, it is created.

(a) `INTEGRATE`

7.6.20 **gammaf**

Computes the Gamma function $\Gamma(x)$.

```
gammaf(x)
```

$$\int_0^\infty t^{x-1} \cdot e^{-t}\, dt$$

(b) `gammaf`

7.7.1 **derivative**

Computes the derivative of the expression $f(x)$ given in the first argument with respect to the variable x given in the second argument at the point $x = a$ given in the third argument using an adaptive scheme. The fourth optional argument h is the initial width of the range the adaptive derivation method starts with. The fifth optional argument p is a flag that indicates whether a backward ($p < 0$), centered ($p = 0$) or forward ($p > 0$) stencil is to be used. This functional calls the GSL functions `gsl_deriv_backward`, `gsl_deriv_central` or `gsl_deriv_forward` according to the indicated flag p. Defaults are $h = (1/2)^{-10} \approx 9.8 \times 10^{-4}$ and $p = 0$.

```
derivative(f(x), x, a, [h], [p])
```

$$\frac{d}{dx}\big[f(x)\big]\Big|_{x=a}$$

(c) `derivative`

7.5.5.2 **keff**

The effective multiplication factor k_{eff}.

7.5.5.3 **sn_alpha**

The stabilization parameter α for S_N.

(d) `keff` y `sn_alpha`

Figura 4.15.: Documentación en PDF a partir de comentarios especiales y meta-datos en el código fuente.

```
///kw_pde+INTEGRATE+detail If the integrand is a function, do not include the
    arguments, i.e. instead of `f(x,y,z)` just write `f`.
///kw_pde+INTEGRATE+detail The results should be the same but efficiency will be
    different (faster for pure functions).
  char *token = feenox_get_next_token(NULL);
  if ((mesh_integrate->function = feenox_get_function_ptr(token)) == NULL) {
    feenox_call(feenox_expression_parse(&mesh_integrate->expr, token));
  }
```

- funciones internas

```
///fn+gammaf+desc Computes the Gamma function $\Gamma(x)$.
///fn+gammaf+usage gammaf(x)
///fn+gammaf+math \int_0^\infty t^{x-1} \cdot e^{-t} \, dt
///fn+gammaf+plotx  1 5 1e-1   1 5 1   0 25 5  0.5 2.5
double feenox_builtin_gammaf(expr_item_t *f) {
  double x = feenox_expression_eval(&f->arg[0]);
  return (x <= 0) ? 1 : gsl_sf_gamma(x);
}
```

- funcionales

```
///fu+derivative+usage derivative(f(x), x, a, [h], [p])
///fu+derivative+desc Computes the derivative of the expression $f(x)$
///fu+derivative+desc given in the first argument with respect to the variable $x$
///fu+derivative+desc given in the second argument at the point $x=a$ given in
///fu+derivative+desc the third argument using an adaptive scheme.
///fu+derivative+desc The fourth optional argument $h$ is the initial width
```

```
///fu+derivative+desc of the range the adaptive derivation method starts with.
///fu+derivative+desc The fifth optional argument $p$ is a flag that indicates
///fu+derivative+desc whether a backward ($p < 0$), centered ($p = 0$) or forward ($p > 0$)
///fu+derivative+desc stencil is to be used.
///fu+derivative+desc This functional calls the GSL functions
///fu+derivative+desc `gsl_deriv_backward`, `gsl_deriv_central` or `gsl_deriv_forward`
///fu+derivative+desc according to the indicated flag $p$.
///fu+derivative+desc Defaults are $h = (1/2)^{-10} \approx 9.8 \times 10^{-4}$ and $p = 0$.
///fu+derivative+math \left. \frac{d}{dx} \Big[ f(x) \Big] \right|_{x = a}
double feenox_builtin_derivative(expr_item_t *a, var_t *var_x) {
```

- variables especiales de S_N

```
///va_neutron_sn+keff+desc The effective multiplication factor\ $k_\text{eff}$.
  neutron_sn.keff = feenox_define_variable_get_ptr("keff");

///va_neutron_sn+sn_alpha+desc The stabilization parameter\ $\alpha$ for $S_N$.
  neutron_sn.sn_alpha = feenox_define_variable_get_ptr("sn_alpha");
  feenox_var_value(neutron_sn.sn_alpha) = 0.5;
```

dan lugar a la documentación final en PDF ilustrada en la figura 4.15.

Como algunos párrafos de documentación aparecen en más de un único lugar en la documentación entonces hay un esquema de inclusión de archivos de Markdown manejado por un filtros escritos en Lua. Un pequeño ejemplo puede aparecer

- en el README
- en la descripción en Texinfo
- en el SDS
- en el manual de referencia
- etc.

Por ejemplo, las instrucciones para clonar el repositorio y hacer el bootstrapping, la configuración y la compilación están en

- el README principal (que a su vez es el index de la página web),
- en las instrucciones de compilación detalladas, y
- en la sección de “downloads” de la página web.

Observación. La documentación de FeenoX se distribuye bajo los términos de la GNU Free Documentation License v1.3, o cualquier versión posterior.

5. Resultados

> FeenoX has releases with a proper tarball! It has `INSTALL`, `./configure` and just compiles. Wow. Yeah, these are free software basics, but majority of the sim software (some discrete circuit simulators included!) I've tried fail on most of these points. So thanks for making quality software!
> *Tibor 'Igor2' Palinkas, maintainer of the open source PCB editor pcb-rnd*

> Es el "tocate una que sepamos todos" de S_N.
> *Dr. Ignacio Márquez Damián, sobre el problema de Reed (2023)*

En este capítulo mostramos diez problemas resueltos con la herramienta computacional FeenoX descripta en el capítulo 4 que ilustran algunas de sus características particulares. Cada uno de estos diez problemas no puede ser resuelto con un solver neutrónico a nivel núcleo que no soporte alguno de los cuatro puntos distintivos de FeenoX:

a. Filosofía Unix, integración en scripts y simulación programática
b. Mallas no estructuradas
c. Ordenadas discretas (además de difusión)
d. Paralelización en varios nodos de cálculo con MPI

Problema	a	b	c	d
Mapeo en mallas no conformes (5.1)	●	●		◒
El problema de Reed (5.2)	○	◒	●	
Benchmark PWR IAEA (5.3)	◒	●		◒
El problema de Azmy (5.4)	●	●	●	○
Benchmarks de Los Alamos (5.5)	●	◒	●	
Slab a dos zonas (5.6)	●	●		
Reactor cubo-esfera (5.7)	●	●		
El problema de los pescaditos (5.8)	●	●	○	
MMS con el Stanford bunny (5.9)	●	●	○	○
PHWR vertical con barras inclinadas (5.10)	●	●	●	●

●	requerido	◒	recomendado	○	opcional

Observación. Excepto el primer problema de la sección 5.1, este capítulo se centra en resolver neutrónica a nivel de núcleo tanto con difusión como con ordenadas discretas. En el documento Software Design Specification del apéndice B se pueden encontrar otro tipo de problemas que ilustran cómo FeenoX resuelve los requerimientos del apéndice A. En la página web de FeenoX, en particular en el apartado de ejemplos, se pueden encontrar más problemas resueltos divididos por tipo de ecuación en derivadas parciales (conducción de calor, elasticidad lineal, análisis modal, neutrónica por difusión, neutrónica por ordenadas discretas).

Observación. Todos los archivos necesarios para reproducir los resultados mostrados en este capítulo, junto con el fuente original de esta tesis en Markdown y los archivos de metadata necesarios para compilar a PDF y/o HTML a través de LaTeX, están disponibles en https://github.com/[redacted]thesis bajo licencia CC-BY. Todas las herramientas utilizadas, incluyendo el sistema operativo, el mallador, el propio solver FeenoX, los post-procesadores, los graficadores, los generadores de documentación (y todas las bibliotecas de las cuales todo este software depende) son libres y/o de código abierto.

Observación. Las mallas de los problemas resueltos en este capítulo (incluso la malla uniforme del caso i de la sección 5.6) han sido generadas con la herramienta de mallado Gmsh [20] y las vistas de post-procesamiento han sido creadas con la herramienta ParaView [1]. Ambas son libres, abiertas y amenas a la simulación programática.

5.1. Mapeo en mallas no conformes

> **TL;DR**: Sobre la importancia de que FeenoX siga la filosofía Unix.

Este primer caso no resuelve ninguna PDE pero sirve para ilustrar…

1. las ideas de la filosofía Unix [47], [50], en particular programas que…
   - hagan una cosa y que la hagan bien
   - trabajen juntos
   - manejen flujos[1] de texto porque esa es una interfaz universal.
2. la capacidad de FeenoX de leer distribuciones espaciales definidas sobre los nodos de una cierta malla no estructurada y de evaluarla en posiciones **x** arbitrarias.

Una aplicación de esta segunda característica es leer una distribución espacial de temperaturas calculadas por un solver térmico (el mismo FeenoX podría servir) y utilizarlas para construir la matriz de rigidez de otro problema (por ejemplo elasticidad lineal para problemas termo-mecánicos o transporte o difusión de neutrones para neutrónica realimentada con termohidráulica). En este caso, los puntos de evaluación son los puntos de Gauss de los elementos de la segunda malla.

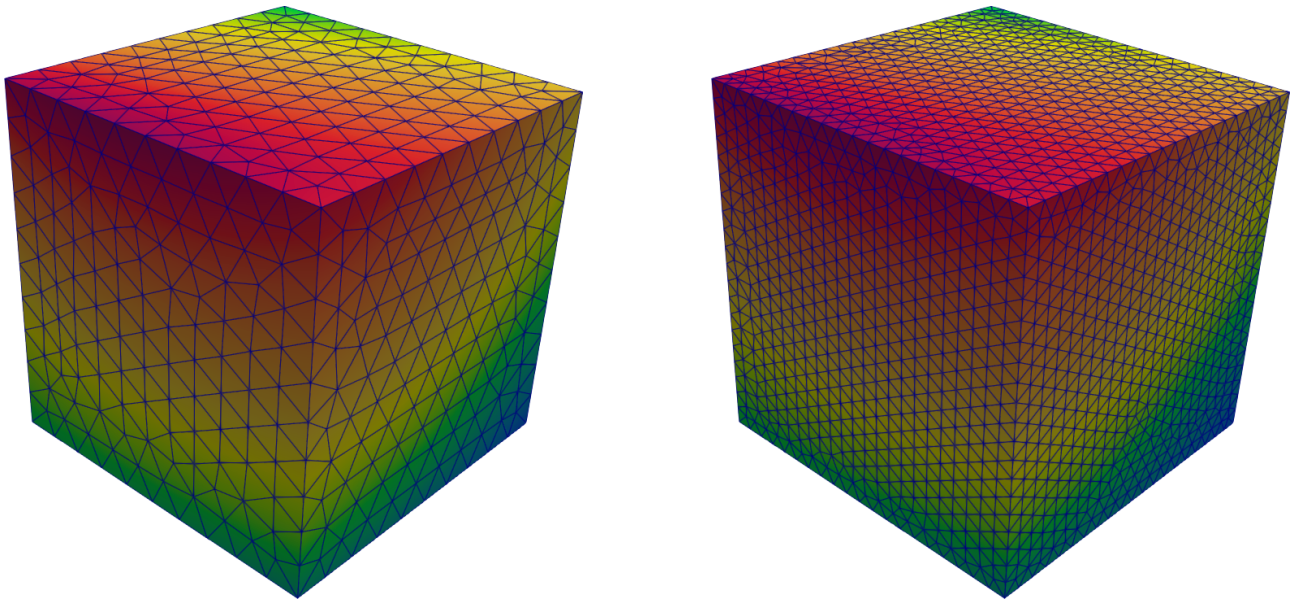
En este problema comenzamos escribiendo una función $f(x, y, z)$ definida algebráicamente en los nodos de un cubo unitario $[0, 1] \times [0, 1] \times [0, 1]$ creado en Gmsh con la instrucción `Box` que llama a la primitiva apropiada del núcleo[2] OpenCASCADE:

[1] Del inglés *streams*.
[2] En el sentido del inglés *kernel*.

```
SetFactory("OpenCASCADE");
Box(1) = {0, 0, 0, 1, 1, 1};
```

Realizamos el mallado con un algoritmo completamente no estructurado utilizando una cierta cantidad n_1 de elementos por lado. Luego, leemos esa malla de densidad c_1 con los valores nodales de $f(\mathbf{x})$ y los interpolamos en la posición de los nodos del mismo cubo mallado con otra densidad n_2. Como hemos partido de una función algebraica, podemos evaluar el error cometido en la interpolación en función de las densidades n_1 y n_2.

Observación. Este procedimiento no es exactamente el necesario para realizar cálculos acoplados ya que la evaluación en la segunda malla es sobre los nodos y no sobre los puntos de Gauss, pero el concepto es el mismo: interpolar valores nodales en puntos arbitrarios.



(a) $n = 10$ (b) $n = 20$

Figura 5.1.: Función $f(\mathbf{x})$ evaluada en el cubo unitario para dos diferentes mallas

El script `run.sh` realiza una inicialización (0) y tres pasos (1–3):

0. Lee como un string el primer argumento `$1` en la línea de comandos después del archivo de entrada[3] la función $f(x, y, z)$. Si no se provee ningún argumento, utiliza como default

$$f(x, y, z) = 1 + x \cdot \sqrt{y} + 2 \cdot \log(1 + y + z) + \cos(xz) \cdot e^{y \cdot z}$$

1. Llama a Gmsh con el archivo de entrada `cube.geo` para crear cinco mallas con $n = 10, 20, 30, 40, 50$ elementos por lado a partir del cubo base con diferentes valores del parámetro `-clscale`:

[3]Notar que el hecho de que los argumentos estén disponibles en el input como `$1`, `$2`, etc. (o incluso como `${1}`, `${2}`, etc.) coincide con la sintaxis de Bash, lo que sigue la regla de Unix de sorpresa mínima (sección C.10).

```
SetFactory("OpenCASCADE");
Box(1) = {0, 0, 0, 1, 1, 1};
```

Cada una de estas cinco mallas `cube-n.msh` (donde `n` es 10, 20, 30, 40 o 50) es leída por FeenoX y se crea un archivo nuevo llamado `cube-n-src.msh` con un campo escalar `f` definido sobre los nodos según el argumento pasado por `run.sh` a FeenoX en `$1` (figura 5.1):

```
READ_MESH cube-$2.msh
f(x,y,z) = $1
WRITE_MESH cube-$2-src.msh f
```

2. Para cada combinación $n_1 = 10, \ldots, 50$ y $n_2 = 10, \ldots, 50$, lee la malla `cube-n1-src.msh` con el campo escalar `f` y define una función $f(x,y,z)$ definida por puntos en los nodos de la malla de entrada. Entonces escribe un archivo de salida VTK llamado `cube-n1-n2-dst.vtk` con dos campos escalares nodales sobre la malla de salida `cube-$3.msh`:

    1. la función $f(x,y,z)$ de la malla de entrada interpolada en la malla de salida
    2. el valor absoluto de la diferencia entre la $f(x,y,z)$ interpolada y la expresión algebraica original de referencia:

```
READ_MESH cube-$2-src.msh DIM 3 READ_FUNCTION f
READ_MESH cube-$3.msh
WRITE_MESH cube-$2-$3-dst.vtk f NAME error "abs(f(x,y,z)-($1))" MESH cube-$3.msh
```

3. Finalmente, para cada archivo VTK, lee el campo escalar como `f_msh` y calcula el error L_2 como

$$e_2 = \int \sqrt{\left[f_{\text{msh}}(\mathbf{x}) - f(\mathbf{x})\right]^2}\, d^3\mathbf{x}$$

y el error L_∞ como

$$e_\infty = \max \left| f_{\text{msh}}(\mathbf{x}) - f(\mathbf{x}) \right|$$

e imprime una línea con un formato adecuado para que el script `run.sh` pueda escribir una tabla Markdown que pueda ser incluida en un archivo de documentación con control de versiones Git, tal como esta tesis de doctorado.

```
READ_MESH cube-$2-src.msh DIM 3 READ_FUNCTION f
READ_MESH cube-$3.msh
WRITE_MESH cube-$2-$3-dst.vtk f NAME error "abs(f(x,y,z)-($1))" MESH cube-$3.msh
```

La tabla 5.3 muestra el tiempo necesario para generar lo datos. La tabla 5.4 muestra los errores y el tiempo necesario para interpolar los datos.

Observación. El cálculo del error L_∞ se hace sobre los nodos y sobre los puntos de Gauss. Recordar la figura 4.4.

n	elementos	nodos	tiempo de mallado [s]	tiempo de rellenado [s]
10	4.979	1.201	0,09	0,01
20	37.089	7.411	0,41	0,05
30	123.264	22.992	1,19	0,26
40	289.824	51.898	3,23	0,87
50	560.473	98.243	7,04	1,85

(a)

Tabla 5.3.: Tiempo necesario para generar los datos.

n_1	n_2	error L_2	error L_∞	tiempo [s]
10	10	1.3×10^{-2}	6.2×10^{-6}	0.02
10	20	1.3×10^{-2}	9.0×10^{-2}	0.08
10	30	1.3×10^{-2}	9.6×10^{-2}	0.32
10	40	1.3×10^{-2}	9.4×10^{-2}	1.01
10	50	1.3×10^{-2}	9.8×10^{-2}	1.94
20	10	1.3×10^{-2}	4.1×10^{-3}	0.06
20	20	6.2×10^{-3}	6.9×10^{-6}	0.11
20	30	6.4×10^{-3}	6.4×10^{-2}	0.40
20	40	6.2×10^{-3}	6.7×10^{-2}	0.98
20	50	6.1×10^{-3}	6.7×10^{-2}	2.30
30	10	1.3×10^{-2}	1.7×10^{-3}	0.29
30	20	6.4×10^{-3}	6.4×10^{-3}	0.36
30	30	4.2×10^{-3}	7.1×10^{-6}	0.57
30	40	4.3×10^{-3}	4.7×10^{-2}	1.48
30	50	4.2×10^{-3}	5.3×10^{-2}	2.78
40	10	1.3×10^{-2}	1.2×10^{-3}	0.99
40	20	6.3×10^{-3}	5.3×10^{-3}	1.06
40	30	4.3×10^{-3}	1.3×10^{-2}	1.44
40	40	3.1×10^{-3}	7.4×10^{-6}	1.95
40	50	3.2×10^{-3}	3.6×10^{-2}	3.79
50	10	1.3×10^{-2}	6.0×10^{-4}	2.07
50	20	6.2×10^{-3}	2.1×10^{-3}	2.31
50	30	4.2×10^{-3}	3.9×10^{-3}	2.62
50	40	3.2×10^{-3}	2.4×10^{-2}	3.74
50	50	2.5×10^{-3}	7.3×10^{-6}	4.26

(a)

Tabla 5.4.: Errores y tiempos necesarios para interpolar los datos.

	Otro	FeenoX
Tiempo	33.4 seg	7.24 seg
Error L_2	2.859×10^{-5}	2.901×10^{-5}
Dif. más negativa	-2.509×10^{-4}	-5.544×10^{-3}
Dif. más positiva	$+1.477 \times 10^{-4}$	$+7.412 \times 10^{-4}$

(a) De $n_1 = 50$ (98.243 nodos) a $n_2 = 100$ (nodos 741.243 nodos)

	Otro	FeenoX
Tiempo	54.2 seg	1.63 seg
Error L_2	6.937×10^{-6}	6.797×10^{-6}
Dif. más negativa	-6.504×10^{-5}	-5.164×10^{-5}
Dif. más positiva	$+2.605 \times 10^{-5}$	$+3.196 \times 10^{-5}$

(b) De $n_1 = 100$ (741.243 nodos) a $n_2 = 50$ (98.243 nodos)

Tabla 5.5.: Comparación de tiempos de mapeo entre FeenoX y otra alternativa

Observación. Si $f(\mathbf{x})$ fuese lineal o incluso polinómica, los errores serían mucho menores.

Para finalizar este primer caso, las tablas 5.5a y 5.5b muestran los errores y los tiempos necesarios para realizar el mismo mapeo entre FeenoX y una biblioteca que forma parte de una solución comercial[4] vendida por unas de las empresas de software de elementos finitos con mayor participación el el mercado mundial.

En el repositorio https://github.com/[redacted]/feenox-non-conformal-mesh-interpolation se pueden encontrar más detalles sobre el análisis del mapeo no conforme propuesto por FeenoX.

5.2. El problema de Reed

> **TL;DR**: Este problema tiene más curiosidad histórica que numérica. Es uno de los problemas más sencillos no triviales que podemos encontrar y sirve para mostrar que para tener en cuenta regiones vacías no se puede utilizar una formulación de difusión.

Este caso, que data de 1971 [48], es de los más sencillos que FeenoX puede resolver. Por lo tanto, por base de diseño, el archivo de entrada también debe ser sencillo. Aprovechamos, entonces, la sencillez de este primer problema para explicar en detalle cómo generar la malla con Gmsh y cómo preparar este archivo de entrada en forma apropiada.

El problema de Reed consiste en una geometría tipo slab adimensional para $0 < x < 8$ con cinco zonas, cada una con secciones eficaces macroscópicas adimensionales uniformes (figura 5.2):

[4] El término "comercial" no está siendo usado como oposición a "software libre" o "código abierto". Como discutimos en la sección 4.4.1, es éste un error común. Pero de ninguna manera que un software sea comercial implica que no pueda ser libre o abierto. La palabra "comercial" solamente indica que la herramienta con la que comparamos FeenoX forma parte de una biblioteca que se vende comercialmente, hay clientes que pagan por usarla y hay personas que dan soporte técnico a los clientes como un servicio de post-venta.

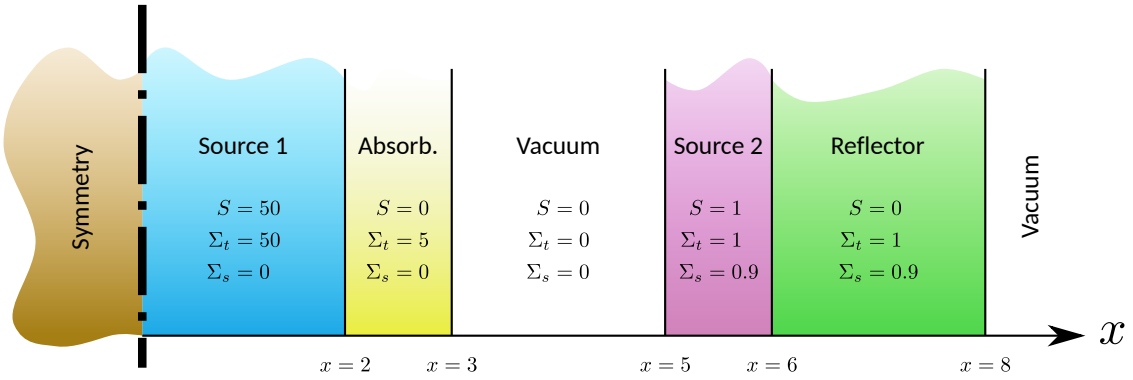


Figura 5.2.: El problema de Reed [48]

a. $0 < x < 2 \mapsto$ Source 1
b. $2 < x < 3 \mapsto$ Absorber
c. $3 < x < 5 \mapsto$ Vacuum
d. $5 < x < 6 \mapsto$ Source 2
e. $6 < x < 8 \mapsto$ Reflector

- Las fuentes de neutrones son independientes y no hay materiales físiles (ni fisionables), por lo que el problema a resolver es un sistema lineal de ecuaciones (KSP) y no un problema de autovalores (EPS)
- El material "vacuum´" tiene secciones eficaces nulas, lo que implica que no puede utilizarse la aproximación de difusión ya que el coeficiente $D(x)$ estaría mal definido.
- Se espera que haya gradientes espaciales grandes en las interfaces entre materiales, por lo que vamos a refinar localizadamente alrededor de los puntos $x = 2$, $x = 3$, $x = 5$ y $x = 6$.
- Como mencionamos en la definición 3.26, la ecuación de transporte es hiperbólica y necesita un término de estabilización en el término convectivo. FeenoX implementa un método tipo SUPG (definición 3.26) controlado por un factor α que puede ser definido explícitamente en el archivo de entrada a través de la variable especial `sn_alpha`. Por defecto, $\alpha = 1/2$.
- La condición de contorno en $x = 0$ es tipo simetría, lo que implica que FeenoX utilice el método de penalidad para implementarla. Es posible elegir el peso en el archivo de entrada con la variable especial `penalty_weight`. Valores altos implican mayor precisión en la condición de contorno pero peor condicionamiento de la matriz global de rigidez `K`.
- La condición de contorno en $x = 8$ es vacío, lo que corresponde a una condición de Dirichlet para las direcciones entrantes.

Podemos generar la geometría y la malla del problema `reed.msh` (que luego será leída por FeenoX) con el siguiente archivo de entrada de Gmsh:

```
//
//     |          |    |         |    |         |
//  m  | src= 50  | 0  |    0    | 1  |    0    |    v
//  i  |          |    |         |    |         |    a
//  r  | tot= 50  | 5  |    0    | 1  |    1    |    c
//  r  |          |    |         |    |         |    u
//  o  | scat=0   | 0  |    0    | 0.9|   0.9   |    u
//  r  |          |    |         |    |         |    m
//     |          |    |         |    |         |
//     |    1     | 2  |    3    | 4  |    5    |
//     |          |    |         |    |         |
```

```
//     +---------+----+---------+----+---------+-------> x
//    x=0       x=2  x=3       x=5  x=6       x=8

lc0 = 0.25;   // tamaño de elemento base
f = 0.1;      // factor de refinamiento

// puntos en extremos de interfaces
Point(1) = {0, 0, 0, lc0};
Point(2) = {2, 0, 0, f*lc0};
Point(3) = {3, 0, 0, f*lc0};
Point(4) = {5, 0, 0, f*lc0};
Point(5) = {6, 0, 0, f*lc0};
Point(6) = {8, 0, 0, lc0};

// puntos medios de cada zona
Point(11) = {1,   0, 0, lc0};
Point(12) = {2.5, 0, 0, lc0};
Point(13) = {4,   0, 0, lc0};
Point(14) = {5.5, 0, 0, lc0};
Point(15) = {7,   0, 0, lc0};

// líneas geométricas
Line(1) = {1, 11};
Line(11) = {11, 2};
Line(2) = {2, 12};
Line(12) = {12, 3};
Line(3) = {3, 13};
Line(13) = {13, 4};
Line(4) = {4, 14};
Line(14) = {14, 5};
Line(5) = {5, 15};
Line(15) = {15, 6};

// líneas físicas
Physical Line("source1")   = {1,11};
Physical Line("absorber")  = {2,12};
Physical Line("void")      = {3,13};
Physical Line("source2")   = {4,14};
Physical Line("reflector") = {5,15};

// puntos físicos
Physical Point("left")  = {1};
Physical Point("right") = {6};
```

lo que da lugar a 81 nodos distribuidos heterogéneamente como ilustramos en la figura 5.3.

Figura 5.3.: 81 nodos y 80 elementos tipo línea para el problema de Reed.

Estamos entonces en condiciones de preparar el archivo de entrada para FeenoX:

```
# ordenadas discretas en una dimensión a un grupo de energías
# leer N de la línea de comandos
PROBLEM neutron_sn DIM 1 GROUPS 1 SN $1
```

```
# leer la malla de este archivo
READ_MESH reed.msh

# propiedades de materiales (todas uniformes por zona)
MATERIAL source1       S1=50 Sigma_t1=50 Sigma_s1.1=0
MATERIAL absorber      S1=0  Sigma_t1=5  Sigma_s1.1=0
MATERIAL void          S1=0  Sigma_t1=0  Sigma_s1.1=0
MATERIAL source2       S1=1  Sigma_t1=1  Sigma_s1.1=0.9
MATERIAL reflector     S1=0  Sigma_t1=1  Sigma_s1.1=0.9

# condiciones de contorno
BC left  mirror
BC right vacuum

# resolver el problema = construir matrices y resolver el sistema
SOLVE_PROBLEM

# escribir la funcion phi1(x) en dos columnas ASCII
PRINT_FUNCTION phi1
```

Este (sencillo) archivo de entrada tiene 6 secciones bien definidas:

1. Definición (`PROBLEM` es un sustantivo) de

   a. el tipo de PDE a resolver (`neutron_sn`)
   b. la dimensión del dominio (`DIM 1`)
   c. la cantidad de grupos de energía (`GROUPS 1`)
   d. el orden N en S_N (`SN $1`) a leer como el primer argumento en la línea de comando de invocación del ejecutable `feenox` luego del archivo de entrada `reed.fee`.

2. Instrucción para leer la malla (`READ_MESH` es un verbo seguido de un sustantivo).

3. Definición de los nombres y propiedades de los materiales (`MATERIAL` es un sustantivo). Si los nombres de los materiales en el archivo de entrada de FeenoX coinciden con el nombre de las entidades físicas cuya dimensión es la del problema (líneas físicas en este caso unidimensional) entonces éstas se asocian implícitamente a los materiales. En cualquier caso, se puede hacer una asociación explícita con tantas palabras clave `LABEL` como sea necesario para cada material.

4. Definición de condiciones de contorno (`BC` es sigla de *boundary condition* que es un sustantivo adjetivado). De la misma manera, si el nombre de la condición de contorno coincide con el nombre de entidades físicas de dimensión menor a la dimensión del problema en la malla, la asociación se hace implícitamente. En forma similar, se pueden agregar palabras clave `LABEL`.

5. Instrucción para indicar que FeenoX debe resolver el problema (`SOLVE_PROBLEM` es un verbo). En este caso sencillo esta instrucción debe venir luego de leer la malla y antes de escribir el resultado. En casos ligeramente más complejos como estudiamos a continuación donde cambiamos los valores por defecto de las variables `sn_alpha` y `penalty_weight`, la instrucción `SOLVE_PROBLEM` debe venir luego de estas instrucciones de asignación.

6. Instrucción para escribir en la salida estándar una columna con la posición de los nodos (en este caso un único valor para x) y el flujo escalar ϕ evaluado en x. Podríamos haber pedido los flujos angulares ψ_{mg} a continuación para obtener más columnas de datos, pero dado que el parámetro N se lee desde la línea de comandos no podemos saber al momento de preparar

el archivo de entrada cuántos flujos angulares van a estar definidos. Por ejemplo, si `$1` es 2 entonces `psi1.1` y `psi1.2` están definidas pero `psi1.3` no lo estará por lo que la línea

```
PRINT_FUNCTION psi1 psi1.1 psi1.2 psi1.3 psi1.4
```

dará un error de parseo si `$1` es 2 (pero funcionará bien si `$1` es 4 tal como ya explicamos en la página 180). En problemas multidimensionales, la instrucción `WRITE_RESULTS` se hará cargo del problema porque escribirá automáticamente en el archivo de salida (en formato Gmsh o VTK) la cantidad correcta de flujos angulares definidos. Otra forma de tener como salida los flujos angulares es reemplazar la instrucción `PRINT_FUNCTION` por

```
INCLUDE print-$1.fee
```

y preparar diferentes archivos `print-2.fee`, `print-4.fee`, `print-6.fee`, etc. cada uno conteniendo la instrucción `PRINT_FUNCTION` con la cantidad apropiada de argumentos para cada N.

La ejecución propiamente dicha de este problema involucra entonces invocar a Gmsh para generar la malla `reed.msh` a partir de `reed.geo` y luego invocar a FeenoX con el archivo de entrada `reed.fee` y el valor de N deseado a continuación. Como queremos construir un gráfico con el perfil de flujo escalar, redireccionamos cada una de las salidas estándar de las diferentes ejecuciones de FeenoX a diferentes archivos ASCII:

```
$ gmsh -1 reed.geo
Info    : Running 'gmsh -1 reed.geo' [Gmsh 4.12.0-git-01ed7170f, 1 node, max. 1 thread]
Info    : Started on Thu Oct 19 19:31:42 2023
Info    : Reading 'reed.geo'...
Info    : Done reading 'reed.geo'
Info    : Meshing 1D...
Info    : [  0%] Meshing curve 1 (Line)
Info    : [ 10%] Meshing curve 2 (Line)
Info    : [ 20%] Meshing curve 3 (Line)
Info    : [ 30%] Meshing curve 4 (Line)
Info    : [ 40%] Meshing curve 5 (Line)
Info    : [ 50%] Meshing curve 11 (Line)
Info    : [ 60%] Meshing curve 12 (Line)
Info    : [ 70%] Meshing curve 13 (Line)
Info    : [ 80%] Meshing curve 14 (Line)
Info    : [ 90%] Meshing curve 15 (Line)
Info    : Done meshing 1D (Wall 0.0142831s, CPU 0.010774s)
Info    : 81 nodes 91 elements
Info    : Writing 'reed.msh'...
Info    : Done writing 'reed.msh'
Info    : Stopped on Thu Oct 19 19:31:42 2023 (From start: Wall 0.0161955s, CPU 0.010992s)
$ feenox reed.fee 2 > reed-s2.csv
$ feenox reed.fee 4 > reed-s4.csv
$ feenox reed.fee 8 > reed-s8.csv
```

Podemos darle una vuelta de tuerca más a la filosofía Unix y reemplazar las últimas tres llamadas explícitas a feenox por un bucle de Bash. Incluso aprovechamos para ordenar las líneas en orden creciente según la primera columna para poder graficar "con líneas":

```
$ for N in 2 4 8; do feenox reed.fee $N | sort -g > reed-s$N.csv; done
$
```

La figura 5.4 muestra el flujo escalar calculado por FeenoX y una comparación con resultados independientes obtenidos con una implementación de un solver 1D ad-hoc en una herramienta matemática privativa y publicados en un blog académico.[5] Esta solución independiente utiliza una malla uniforme con la misma cantidad (81) de nodos que FeenoX.

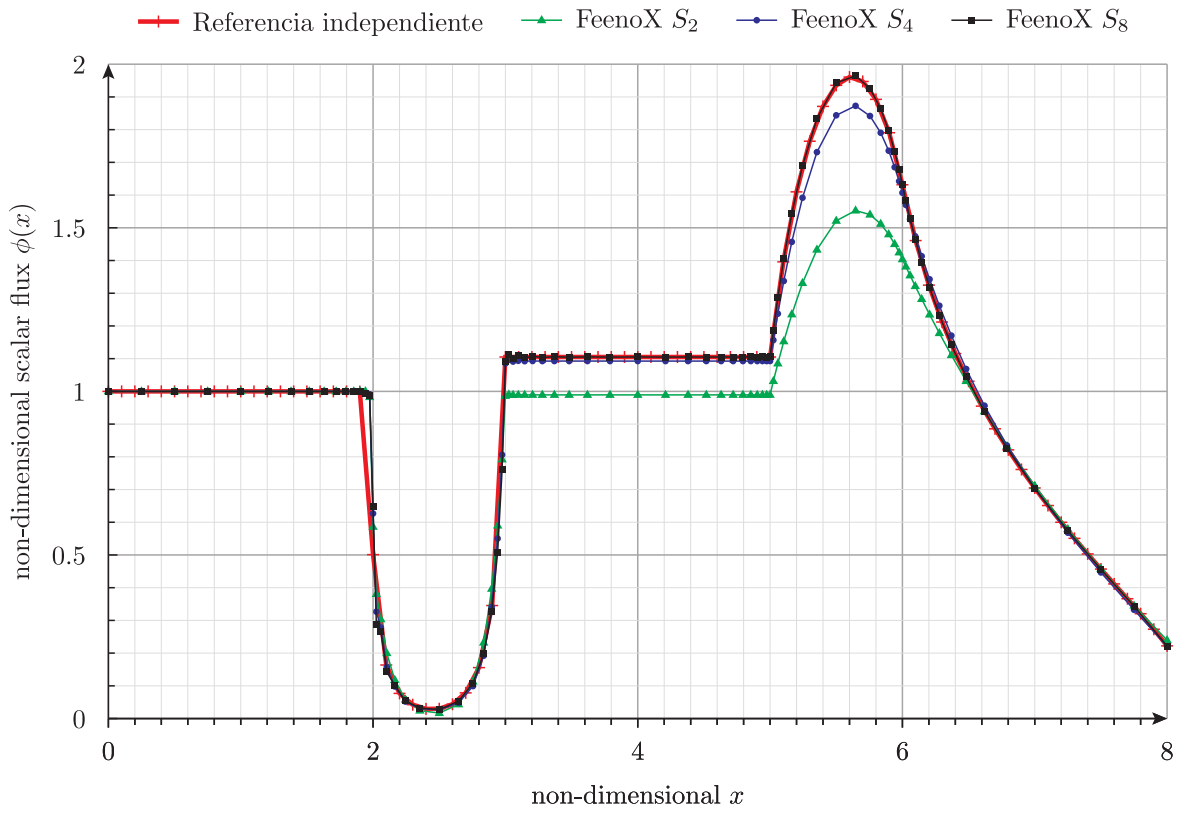


Figura 5.4.: Perfil de flujo angular $\phi(x)$ para el problema de Reed

5.2.1. Efecto del factor de estabilización

Estudiemos brevemente el efecto de modificar el factor de estabilización α mediante la variable `sn_alpha`. Para ello generamos una malla un poco más gruesa con la opción `-clscale` de Gmsh y reducimos el número de nodos de 81 a 53:

```
$ gmsh -1 reed.geo -clscale 2 -o reed-coarse.msh
[...]
Info    : Done meshing 1D (Wall 0.0160784s, CPU 0.011781s)
Info    : 53 nodes 63 elements
Info    : Writing 'reed-coarse.msh'...
Info    : Done writing 'reed-coarse.msh'
Info    : Stopped on Thu Oct 19 20:20:01 2023 (From start: Wall 0.019359s, CPU 0.015746s)
$
```

Agregamos entonces la posibilidad de leer otro argumento en la línea de comandos `$2` y lo asignamos a dicha variable antes de pedir la instrucción `SOLVE_PROBLEM`. Además ahora le pedimos directamente

[5] El autor de dicho blog está al tanto de la comparación con FeenoX.

a FeenoX que escriba la función $\phi_1(x)$ en un archivo de texto cuyo nombre es `$0-$1-$2.csv` donde `$0` es el nombre del archivo de entrada sin la extensión `.fee`:

```
PROBLEM neutron_sn DIM 1 GROUPS 1 SN $1
READ_MESH reed-coarse.msh
MATERIAL source1       S1=50 Sigma_t1=50 Sigma_s1.1=0
MATERIAL absorber      S1=0  Sigma_t1=5  Sigma_s1.1=0
MATERIAL void          S1=0  Sigma_t1=0  Sigma_s1.1=0
MATERIAL source2       S1=1  Sigma_t1=1  Sigma_s1.1=0.9
MATERIAL reflector     S1=0  Sigma_t1=1  Sigma_s1.1=0.9
BC left  mirror
BC right vacuum

sn_alpha = $2

SOLVE_PROBLEM
PRINT_FUNCTION phi1 FILE $0-$1-$2.csv
```

Con dos bucles de Bash anidados probamos todas las combinaciones posibles de $N = 2, 4, 8$ y $\alpha = 0.01, 0.25, 1$:

```
$ for N in 2 4 8; do for alpha in 0.01 0.25 1; do feenox reed-alpha.fee $N $alpha ; done; done
$ for N in 2 4 8; do for alpha in 0.01 0.25 1; do sort -g reed-alpha-$N-$alpha.csv > reed-alpha- ↩
    $N-$alpha-sorted.csv; done; done
```

para obtener la figura 5.5.

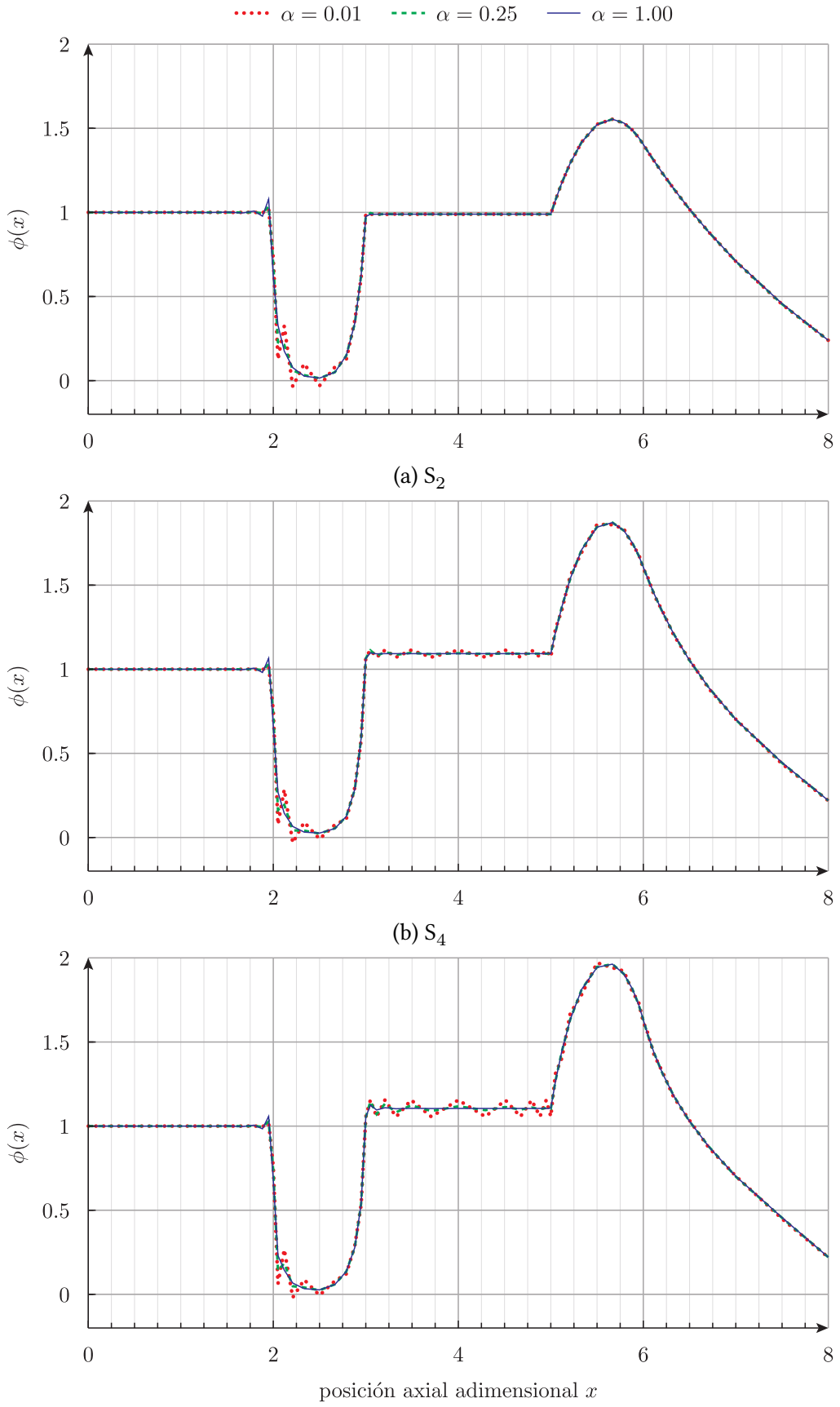
Observación. Como veremos más adelante (por ejemplo en la sección 5.9 o en la sección 5.4), el realizar estudios paramétricos sobre más de un parámetro la cantidad de resultados a analizar aumenta geométricamente. Debido a que FeenoX permite la flexibilidad de ser ejecutado en bucles y de pasar parámetros por líneas de comando, la generación de los resultados es extremadamente eficiente, lo que hace que sea relativamente mucho más difícil el análisis de dichos resultados que la generación de los datos en sí. Esto pone en relieve la importancia de la regla de economía de Unix: no sólo el costo relativo de la unidad de tiempo de CPU es al menos tres órdenes de magnitud menor al costo de la unidad de tiempo de un ingeniero sino que también el tiempo absoluto necesario para analizar resultados es mayor que para generarlos.

5.2.2. Efecto del orden de los elementos

Para finalizar el estudio de este primer problema neutrónico sencillo volvemos a resolver el mismo caso pero utilizando elementos de segundo orden. Está claro que para poder comparar soluciones debemos tener en cuenta el esfuerzo computacional que cada método necesita. Para el mismo tamaño de elemento, el tamaño del problema para una malla de segundo orden es mucho más grande que para una malla de primer orden. Por lo tanto, lo primero que hay que hacer es

a. refinar la malla de primer orden, o
b. hacer más gruesa la malla de segundo orden.

Por otro lado el patrón de la matriz también aumenta (el ancho de banda es mayor en la malla de segundo orden) por lo que también cambia el esfuerzo necesario no sólo para construir la matriz



(a) S_2

(b) S_4

(c) S_8

Figura 5.5.: Efecto del factor de estabilización α

sino también para invertirla, especialmente en términos de memoria. En algunos tipos de problemas (como por ejemplo elasticidad ver sección B.2.2.2), está probado que cualquier esfuerzo necesario para resolver un problema con elementos de segundo orden vale la pena ya que los elementos de primer orden—aún cuando la malla esté muy refinada—padecen del efecto numérico conocido como *shear locking* que arroja resultados poco precisos. Pero en el caso de transporte (e incluso difusión) de neutrones no está claro que, para el mismo tamaño de problema, la utilización de elementos de alto orden sea más precisa que la de elementos de primer orden, más allá de la posibilidad de representar geometrías curvas con más precisión.

De cualquier manera, presentamos entonces resultados para el problema de Reed con elementos unidimensionales de segundo orden. Primeramente le pedimos a Gmsh que nos prepare una malla más gruesa aún que la anterior, pero de orden dos. Esto da 53 nodos, tal como la malla `reed-coarse.msh` de la sección anterior:

```
$ gmsh -1 reed.geo -order 2 -clscale 5 -o reed-coarser2.msh
[...]
Info    : Done meshing order 2 (Wall 0.000161366s, CPU 9.7e-05s)
Info    : 53 nodes 37 elements
Info    : Writing 'reed-coarser2.msh'...
Info    : Done writing 'reed-coarser2.msh'
Info    : Stopped on Fri Oct 20 13:45:05 2023 (From start: Wall 0.0102896s, CPU 0.012253s)
$
```

Ahora preparamos este archivo de entrada que

- utiliza esta malla de segundo orden,
- resuelve el problema de Reed
- lee el flujo obtenido en la sección anterior para $\alpha = 1$, y
- escribe la diferencia algebraica entre los dos flujos escalares en función de x con un paso espacial $\Delta x = 10^{-3}$

para obtener la figura 5.6.

```
PROBLEM neutron_sn DIM 1 GROUPS 1 SN $1
READ_MESH reed-coarser2.msh
MATERIAL source1       S1=50 Sigma_t1=50 Sigma_s1.1=0
MATERIAL absorber      S1=0  Sigma_t1=5  Sigma_s1.1=0
MATERIAL void          S1=0  Sigma_t1=0  Sigma_s1.1=0
MATERIAL source2       S1=1  Sigma_t1=1  Sigma_s1.1=0.9
MATERIAL reflector     S1=0  Sigma_t1=1  Sigma_s1.1=0.9
BC left  mirror
BC right vacuum
sn_alpha = 1
SOLVE_PROBLEM

# leemos la solución de primer orden para comparar
FUNCTION phi_first(x) FILE reed-alpha-$1-1-sorted.csv

# escribimos el flujo y la diferencia
PRINT_FUNCTION phi1 phi_first(x)-phi1(x) MIN 0 MAX 8 STEP 1e-3 FILE $0-$1.csv
```

Observación. No hay una definición o instrucción específica que le indique a FeenoX el orden de los elementos a usar. El solver lee la malla con la instrucción `READ_MESH` y emplea los elementos allí

definidos, que pueden ser de primero o segundo orden. En los casos anteriores, los elementos de mayor orden eran líneas de dos nodos (a.k.a. line2). En este caso, son líneas de tres nodos (a.k.a. line3).

```
$ for N in 2 4 8; do feenox reed2.fee $N; done
$
```

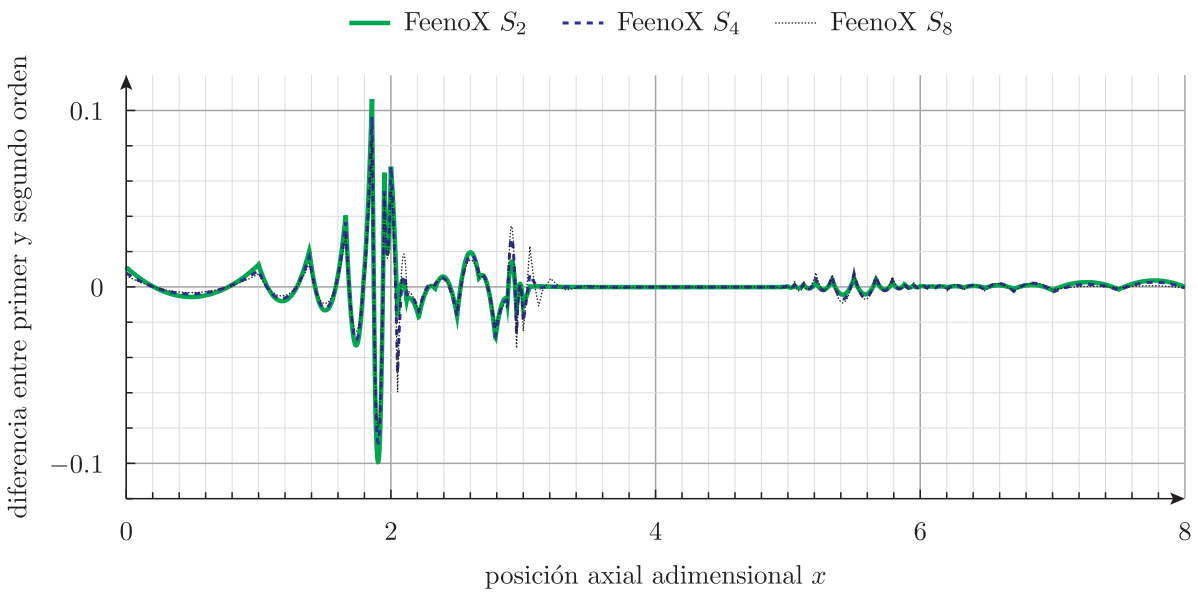


Figura 5.6.: Diferencia entre flujos de primer y segundo orden en el problema de Reed

Observación. Dado que las propiedades de los materiales y las condiciones de contorno fueron siempre iguales para todos los casos resueltos en esta sección, una gestión más eficiente de los archivos de entrada habría implicado que creáramos un archivo separado con las palabras clave `MATERIAL` y `BC` para luego incluir dicho archivo desde cada uno de los archivos de entrada con la palabra clave `INCLUDE` (por ejemplo en la sección 5.10). Como este es el primer problema neutrónico resuelto con FeenoX en esta tesis, hemos elegido dejar explícitamente la definición de materiales y de condiciones de contorno. En secciones siguientes vamos a utilizar la palabra clave `INCLUDE`, que es para lo que fue diseñada.

5.3. IAEA PWR Benchmark

> **TL;DR**: El problema original de 1976 propone resolver un cuarto de núcleo cuando en realidad la simetría es 1/8.

Este problema fue propuesto por Argonne National Laboratory [3] y luego adoptado por la IAEA como un benchmark estándar para validar códigos de difusión. Está compuesto

a. por un problema 2D que representa un cuarto de una geometría típica de PWR sobre el plano x-y más un buckling geométrico para tener en cuenta las pérdidas en la dirección z, y
b. un problema completamente tridimensional de un cuarto de núcleo

Región	D_1	D_2	$\Sigma_{s1\to2}$	Σ_{a1}	Σ_{a2}	$\nu\Sigma_{f2}$	Material
1	1.5	0.4	0.02	0.01	0.08	0.135	Fuel 1
2	1.5	0.4	0.02	0.01	0.085	0.135	Fuel 2
3	1.5	0.4	0.02	0.01	0.13	0.135	Fuel 2 + Rod
4	2.0	0.3	0.04	0	0.01	0	Reflector
5	2.0	0.3	0.04	0	0.055	0	Refl. + Rod

(a) Datos originales de [3] {#tbl-iaea-xs2}. El material 5 es utilizado sólo en el caso 3D.

Tabla 5.6.: Secciones eficaces macroscópicas (uniformes por zonas) del benchmark PWR de IAEA. Al caso 2D se le debe sumar un término de buckling geométrico $B_g^2 = 0.8 \times 10^{-4}$.

5.3.1. Caso 2D original

La figura 5.7, preparada en su momento para la publicación [63], muestra la geometría del problema. La tabla 5.6 muestra las secciones eficaces macroscópicas a dos grupos de cada zona. El problema pide calcular varios puntos, incluyendo

- el factor de multiplicación efectivo k_{eff}
- perfiles de flujo a lo largo de la diagonal
- valores y ubicación de flujos máximos
- potencias medias en cada canal

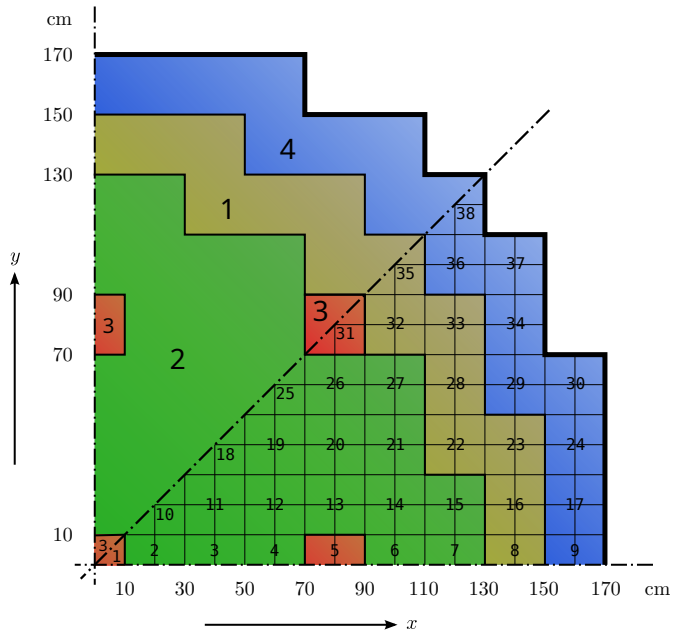


Figura 5.7.: Caso 2D original del benchmark PWR de IAEA

En la referencia [63] hemos resuelto completamente el problema utilizando la segunda versión de la implementación (denominada milonga), incluso utilizando triángulos y cuadrángulos, diferentes algoritmos y densidades de mallado, etc. Más aún, esa versión era capaz de resolver la ecuación de difusión tanto con elementos como con volúmenes finitos tal como explicamos en la monografía [62],

donde también resolvemos el problema. En la presentación [73] mostramos cómo habíamos resuelto el benchmark con la primera versión del código.

En esta sección calculamos solamente el factor de multiplicación y la distribución espacial de flujos. En este caso vamos a prestar más atención al archivo de entrada de FeenoX que a la generación de la malla, que para este caso puede ser estructurada como mostramos en la figura 5.8.

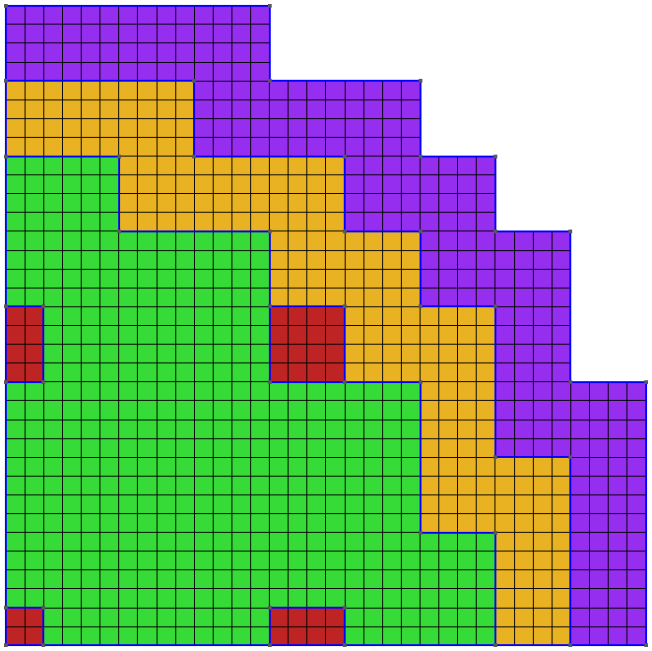


Figura 5.8.: Malla para el caso 2D original con simetría 1/4

```
PROBLEM neutron_diffusion 2D GROUPS 2

DEFAULT_ARGUMENT_VALUE 1 quarter   # quarter o eighth
READ_MESH iaea-2dpwr-$1.msh

Bg2 = 0.8e-4  # buckling geometrico en la dirección z
MATERIAL fuel1 {
  D1=1.5    Sigma_a1=0.010+D1(x,y)*Bg2    Sigma_s1.2=0.02
  D2=0.4    Sigma_a2=0.080+D2(x,y)*Bg2    nuSigma_f2=0.135   }

MATERIAL fuel2 {
  D1=1.5    Sigma_a1=0.010+D1(x,y)*Bg2    Sigma_s1.2=0.02
  D2=0.4    Sigma_a2=0.085+D2(x,y)*Bg2    nuSigma_f2=0.135   }

MATERIAL fuel2rod {
  D1=1.5    Sigma_a1=0.010+D1(x,y)*Bg2    Sigma_s1.2=0.02
  D2=0.4    Sigma_a2=0.130+D2(x,y)*Bg2    nuSigma_f2=0.135   }

MATERIAL reflector {
  D1=2.0    Sigma_a1=0.000+D1(x,y)*Bg2    Sigma_s1.2=0.04
  D2=0.3    Sigma_a2=0.010+D2(x,y)*Bg2 }

BC external vacuum=0.4692
BC mirror   mirror

SOLVE_PROBLEM
```

```
PRINT      "grados de libertad = " total_dofs
PRINT %.5f "keff = " keff
WRITE_RESULTS FORMAT vtk
```

Observación. Hay una relación bi-unívoca bastante clara entre la definición del problema en el reporte [3] y el archivo de entrada necesario para resolverlo con FeenoX. El lector experimentado podrá notar que esta característica (que es parte de la base de diseño del software) no es común en otros solvers, ni neutrónicos ni termo-mecánicos.

Observación. Si bien las secciones eficaces son uniformes, la sección eficaz de absorción está dada por una expresión que es la suma de la sección eficaz base más el producto del coeficiente de difusión D_g por el buckling geométrico B_g. En lugar de volver a escribir la constante numérica correspondiente al material, escribimos $D_g(x,y)$ para que FeenoX reemplace el valor apropiado del coeficiente de difusión del material en cuestión por nosotros.

Observación. Tal como en el problema de Reed de la sección anterior donde teníamos elementos de tipo punto para definir condiciones de contorno, en este caso bi-dimensional la malla contiene elementos de dimensión uno (tipo líneas) donde se aplican las condiciones de contorno. Los elementos que discretizan las líneas $x = 0$ e $y = 0$ tienen asignado (en el archivo de entrada de Gmsh no mostrado) el nombre "vacuum" y los que discretizan el borde externo del reflector, el nombre "mirror". Estos dos nombres son usados en las dos instrucciones BC del archivo de entrada de FeenoX para indicar qué clase de condición de contorno hay que aplicarle a cada grupo de elementos de dimensión topológica menor a la del problema.

```
$ gmsh -2 iaea-2dpwr-quarter.geo
[...]
Info    : Done meshing 2D (Wall 0.0422971s, CPU 0.042152s)
Info    : 1033 nodes 1286 elements
Info    : Writing 'iaea-2dpwr-quarter.msh'...
Info    : Done writing 'iaea-2dpwr-quarter.msh'
Info    : Stopped on Fri Oct 20 15:55:03 2023 (From start: Wall 0.0522626s, CPU 0.060603s)
$ time feenox iaea-2dpwr.fee
grados de libertad =    2066
keff =  1.02985

real    0m0.696s
user    0m0.090s
sys     0m0.162s
$
```

5.3.2. Caso 2D con simetría 1/8

Como deslizamos en el capítulo 1, bien mirado el problema no tiene simetría 1/4 sino simetría 1/8. Sucede que para poder explotar dicha simetría se necesita una malla no estructurada, que ni en 1976 ni en 2024 (excepto algunos casos puramente académicos como [5], [41], [43], [86]) es una característica de los solvers neutrónicos de nivel de núcleo. De hecho el paper [63] justamente ilustra el hecho de que las mallas no estructuradas permiten reducir la cantidad de grados de libertad necesarios para resolver un cierto problema.

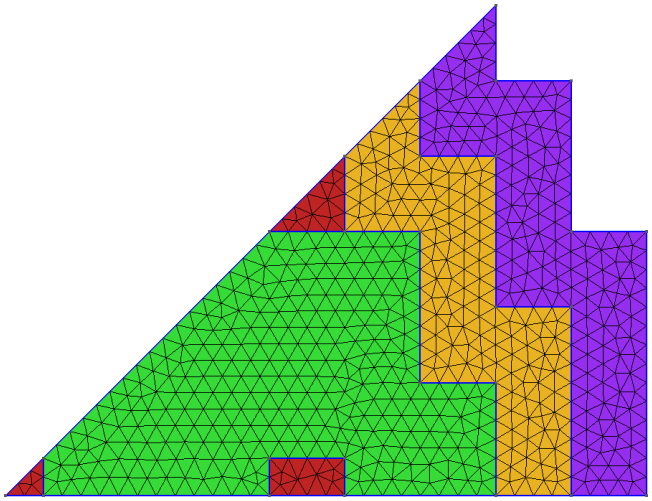


Figura 5.9.: Malla para el caso 2D original con simetría 1/8

Utilizando `eighth` como argumento `$1` podemos usar el mismo archivo de entrada pero con la malla de la figura 5.9:

```
$ gmsh -2 iaea-2dpwr-eighth.geo
[...]
Info    : 668 nodes 1430 elements
Info    : Writing 'iaea-2dpwr-eighth.msh'...
Info    : Done writing 'iaea-2dpwr-eighth.msh'
Info    : Stopped on Fri Oct 20 17:58:06 2023 (From start: Wall 0.0331908s, CPU 0.040327s)
$ time feenox iaea-2dpwr.fee eighth
grados de libertad =    1336
keff =  1.02974

real    0m0.681s
user    0m0.075s
sys     0m0.161s
$
```

Observación. El tiempo de CPU reportado por `time` es el mismo independiente de la cantidad de grados de libertad. Esto indica que el tamaño del problema es muy pequeño y el tiempo necesario para construir las matrices y resolverlas es despreciable frente al overhead de cargar un ejecutable, inicializar bibliotecas compartidas, etc. Podemos verificar esta afirmación analizando la salida de la opción `--log_view` que le indica a PETSc que agregue una salida con datos de performance:

```
$ feenox iaea-2dpwr.fee eighth --log_view
grados de libertad =    1336
keff =  1.02974
[...]
Summary of Stages:   ----- Time ------  ----- Flop ------
                        Avg     %Total     Avg     %Total
 0:      Main Stage: 2.0911e-03   7.7%  0.0000e+00   0.0%
 1:            init: 2.1185e-04   0.8%  0.0000e+00   0.0%
 2:           build: 9.7703e-03  36.1%  0.0000e+00   0.0%
 3:           solve: 1.4487e-02  53.6%  1.9467e+07 100.0%
 4:            post: 4.8677e-04   1.8%  0.0000e+00   0.0%
```

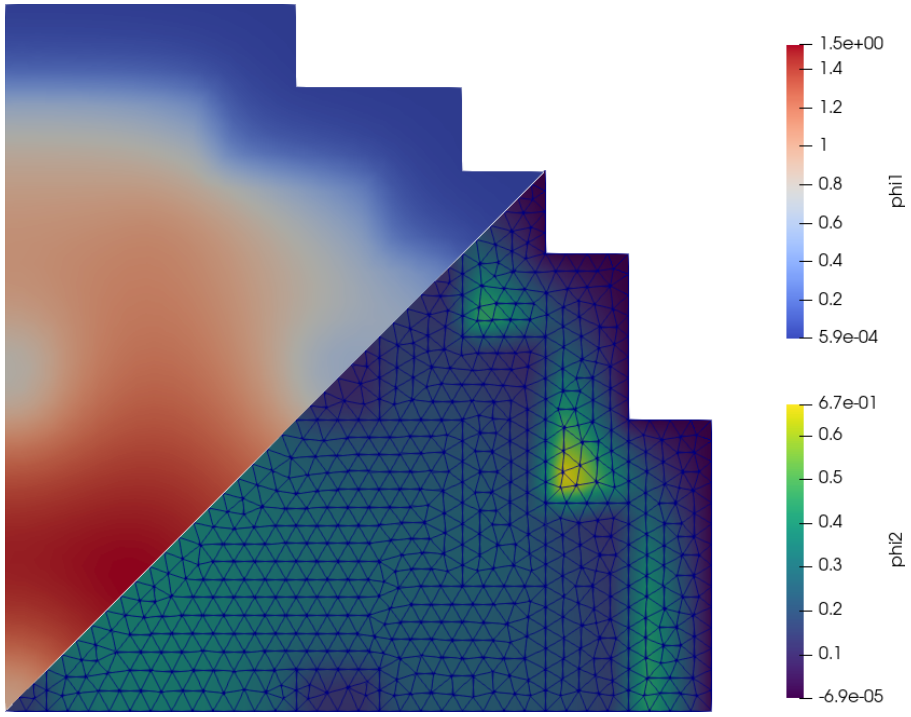


Figura 5.10.: Flujos rápidos y térmicos en el problema 2D de IAEA con simetría 1/8 resuelto con difusión. La imagen del flujo rápido corresponde a una reflexión con respecto a una línea a 45 grados.

```
[...]
$
```

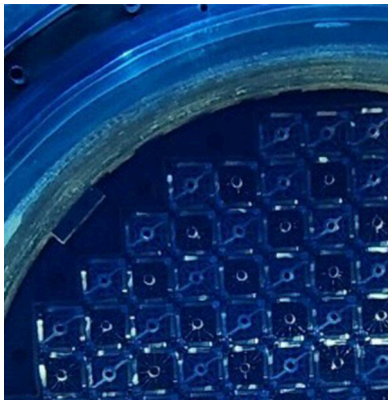
En efecto, se necesitan menos de 10 milisegundos para construir las matrices del problema y menos de 15 para resolverlo.

5.3.3. Caso 2D con reflector circular

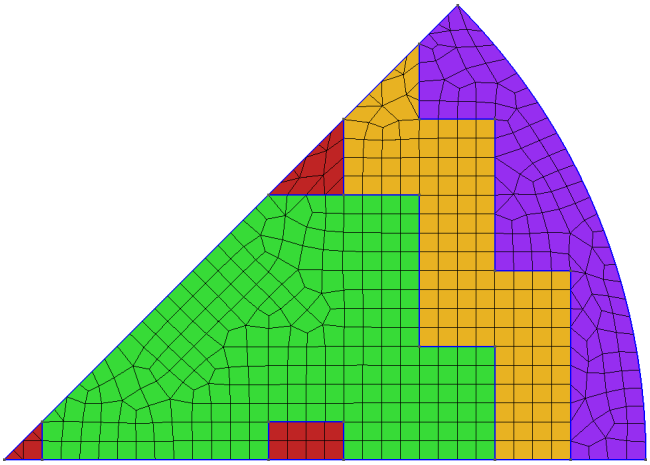
Mirando un poco más en detalle la geometría, hay un detalle que también puede ser considerado con mallas no estructuradas: la superficie exterior del reflector. En efecto, en una geometría tipo PWR cada uno de los canales proyecta un cuadrado en la sección transversal. Pero el reflector sigue la forma del recipiente de presión que es un cilindro (figura 5.11a). Con FeenoX es posible resolver fácilmente esta geometría con el mismo archivo de entrada con la malla de la figura 5.11b que incluye una mezcla de

a. zonas estructuradas y no estructuradas, y
b. triángulos y cuadrángulos.

```
$ gmsh -2 iaea-2dpwr-eighth-circular.geo
Info    : 524 nodes 680 elements
Info    : Writing 'iaea-2dpwr-eighth-circular.msh'...
Info    : Done writing 'iaea-2dpwr-eighth-circular.msh'
Info    : Stopped on Fri Oct 20 17:58:37 2023 (From start: Wall 0.0314288s, CPU 0.043231s)
```



(a) Geometría típica de un PWR



(b) Malla para simetría 1/8 y reflector circular

Figura 5.11.: Un reactor PWR real y un modelo matemático

```
$ time feenox iaea-2dpwr.fee eighth-circular
grados de libertad =    1048
keff =  1.02970

real    0m0.649s
user    0m0.048s
sys     0m0.156s
$
```

5.3.4. Caso 3D con simetría 1/8, reflector circular resuelto con difusión

Pasemos ahora al caso tri-dimensional. El problema original es una extensión sobre el eje z de la geometría con simetría 1/4 y reflector no circular. Como ya vimos en 2D, podemos tener simetría 1/8 y reflector cilíndrico. Ya que estamos en 3D, podemos preparar la geometría con una herramienta tipo CAD como es usual en análisis de ingeniería tipo CAE. En particular, usamos la plataforma CAD Onshape que corre en la nube y se utiliza directamente desde el navegador.[6] La figura 5.12 muestra la geometría continua, que luego de ser mallada con Gmsh con elementos de segundo orden arroja la malla de la figura 5.13.

El archivo de entrada sigue siendo relativamente sencillo, sólo que ahora agregamos un poco más de información a la salida:

```
PROBLEM neutron_diffusion 3D GROUPS 2

DEFAULT_ARGUMENT_VALUE 1 quarter
READ_MESH iaea-3dpwr-$1.msh

MATERIAL fuel1     D1=1.5 D2=0.4 Sigma_s1.2=0.02 Sigma_a1=0.01 Sigma_a2=0.08  nuSigma_f2=0.135
MATERIAL fuel2     D1=1.5 D2=0.4 Sigma_s1.2=0.02 Sigma_a1=0.01 Sigma_a2=0.085 nuSigma_f2=0.135
MATERIAL fuel2rod  D1=1.5 D2=0.4 Sigma_s1.2=0.02 Sigma_a1=0.01 Sigma_a2=0.13  nuSigma_f2=0.135
```

[6] La plataforma CAEplex desarrollada por el autor de esta tesis que provee una interfaz web para una versión anterior de FeenoX corriendo en la nube está 100% integrada en Onshape para realizar cálculos termo-mecánicos.

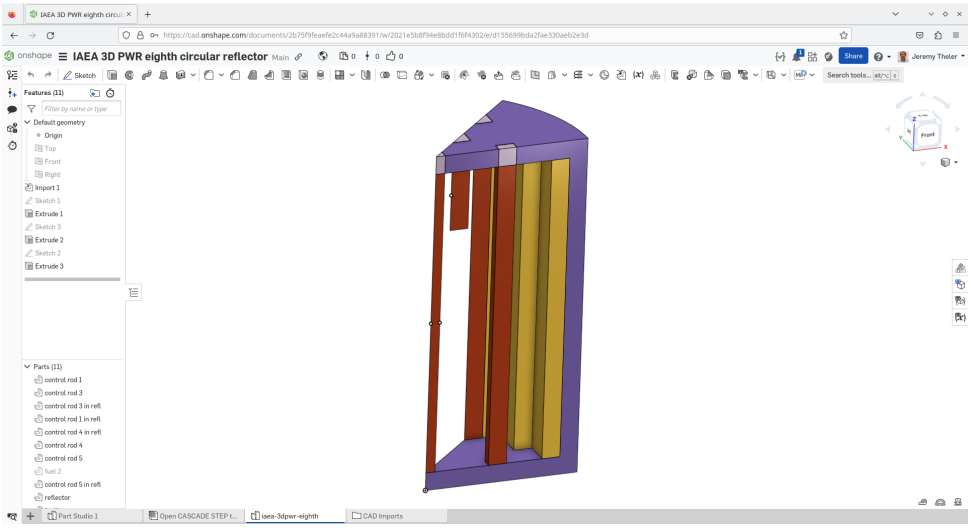
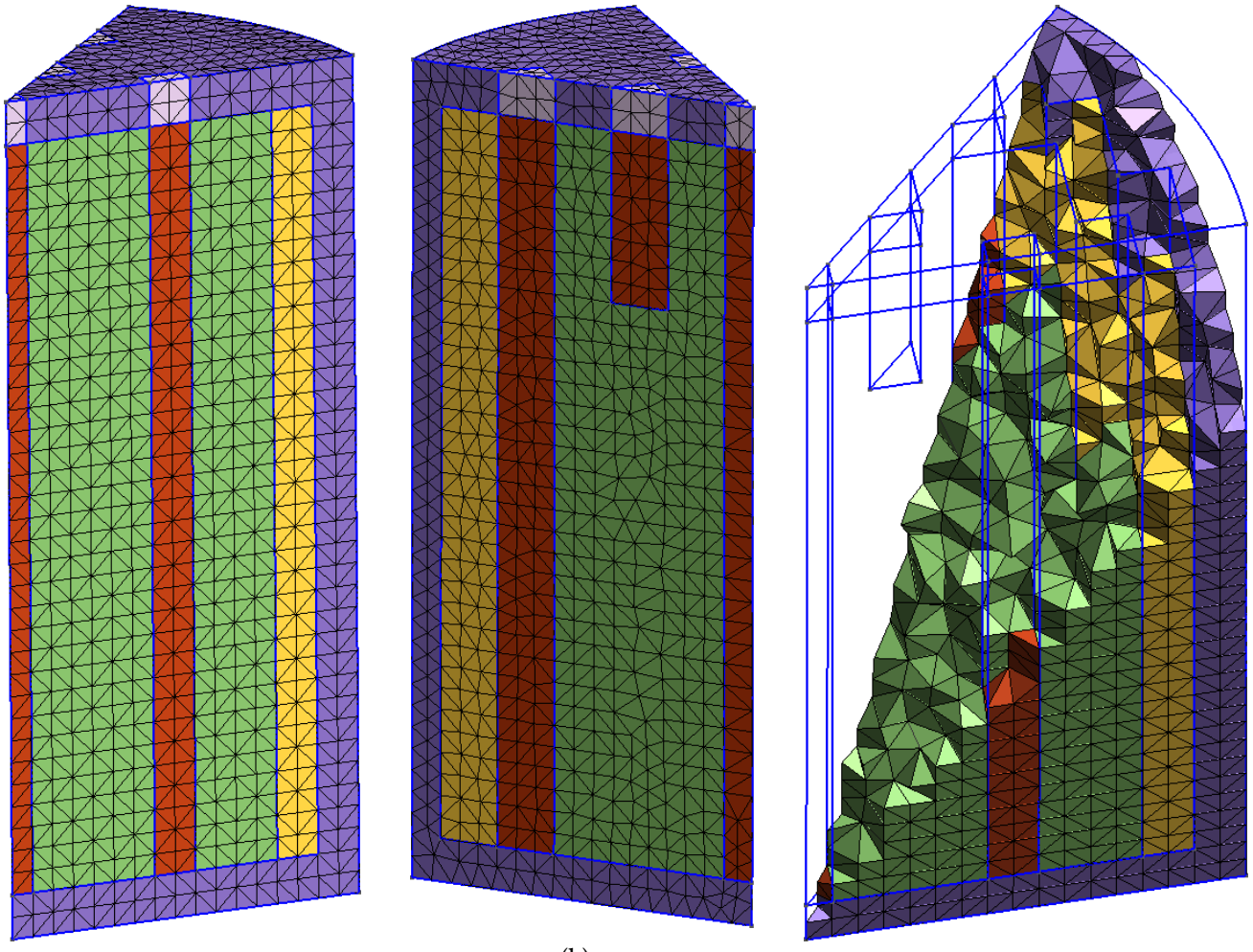


Figura 5.12.: Preparación de la geometría del benchmark PWR 3D de IAEA en Onshape



(a) (b) (c)

Figura 5.13.: Malla para el problema 3D PWR IAEA con simetría 1/8 y reflector circular con elementos tet10 de segundo orden. Cada color indica un material diferente de la tabla 5.6, incluyendo el material número 5 “barras de control en el reflector superior”.

```
MATERIAL reflector D1=2.0 D2=0.3 Sigma_s1.2=0.04 Sigma_a1=0    Sigma_a2=0.01  nuSigma_f2=0
MATERIAL reflrod   D1=2.0 D2=0.3 Sigma_s1.2=0.04 Sigma_a1=0    Sigma_a2=0.055 nuSigma_f2=0

BC vacuum   vacuum=0.4692
BC mirror   mirror

SOLVE_PROBLEM
WRITE_RESULTS FORMAT vtk

PRINT  "geometry = $1"
PRINTF "    keff = %.5f"     keff
PRINTF "   nodes = %g"       nodes
PRINTF "    DOFs = %g"       total_dofs
PRINTF "  memory = %.1f Gb (local) %.1f Gb (global)" mpi_memory_local() mpi_memory_global()
PRINTF "    wall = %.1f sec" wall_time()
```

Como somos ingenieros y tenemos un trauma profesional con el tema de performance, debemos comparar la "ganancia" de usar simetría 1/8 con respecto al original de 1/4:

```
$ feenox iaea-3dpwr.fee quarter
geometry = quarter
    keff = 1.02918
   nodes = 70779
    DOFs = 141558
[0/1 tux]   memory = 2.3 Gb (local) 2.3 Gb (global)
    wall = 26.1 sec
$ feenox iaea-3dpwr.fee eighth
geometry = eighth
    keff = 1.02912
   nodes = 47798
    DOFs = 95596
[0/1 tux]   memory = 1.2 Gb (local) 1.2 Gb (global)
    wall = 12.7 sec
$ feenox iaea-3dpwr.fee eighth-circular
geometry = eighth-circular
    keff = 1.08307
   nodes = 32039
    DOFs = 64078
[0/1 tux]   memory = 0.8 Gb (local) 0.8 Gb (global)
    wall = 7.9 sec
$
```

La figura 5.14 muestra que ahora sí tenemos una ganancia significativa al reducir el tamaño del problema mediante la explotación de la simetría. El tiempo para construir la matriz pasó de 3.2 segundos a 2.0 (recordar que son elementos de segundo orden) y el tiempo necesario para resolver el problema bajó de 21 a 10 segundos.

Podemos investigar un poco qué sucede si quisiéramos resolver el problema en paralelo:

```
$ mpiexec -n 1 feenox iaea-3dpwr.fee quarter
geometry = quarter
    keff = 1.02918
   nodes = 70779
    DOFs = 141558
[0/1 tux]   memory = 2.3 Gb (local) 2.3 Gb (global)
    wall = 26.2 sec
```

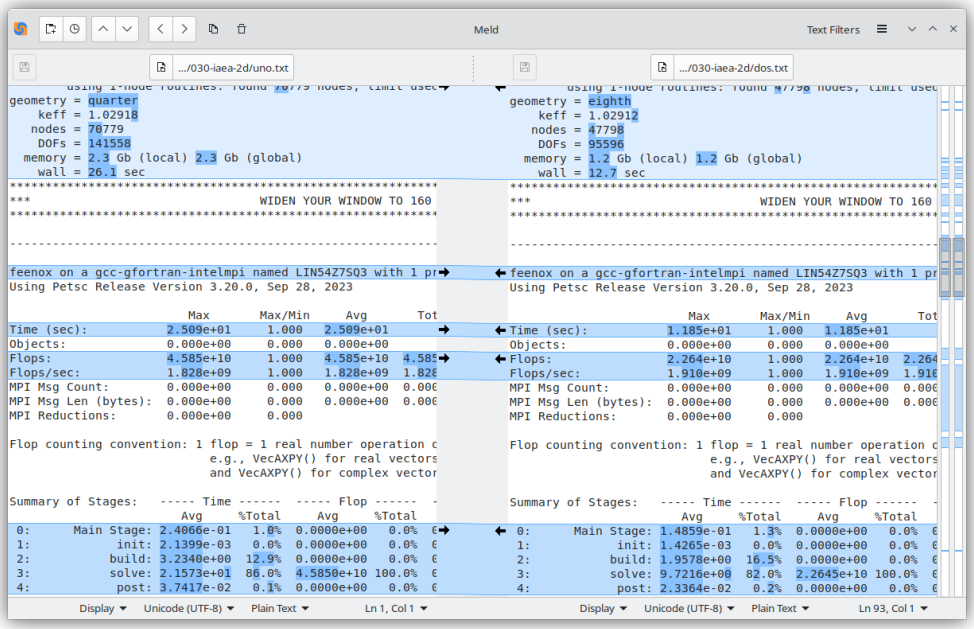


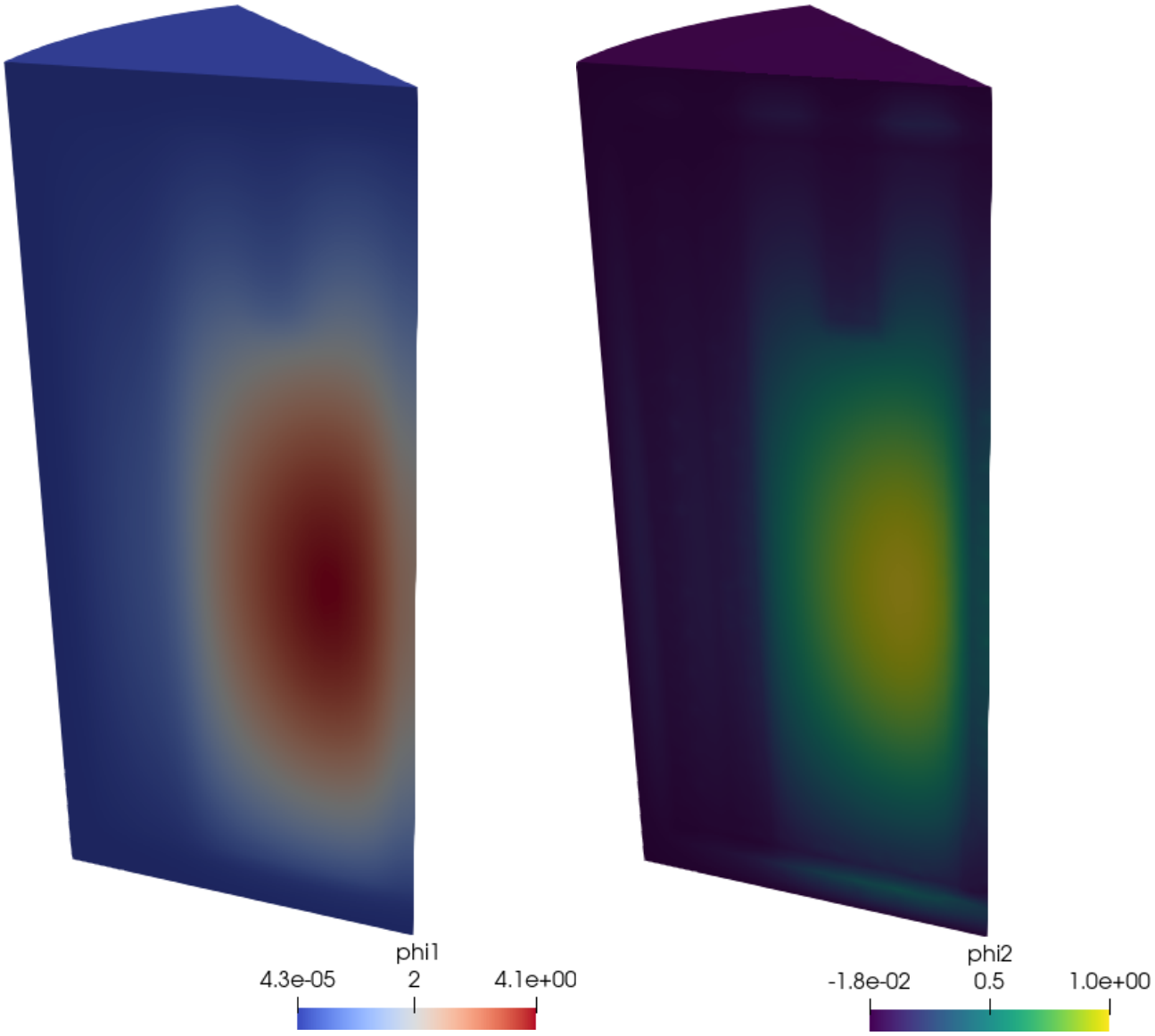
Figura 5.14.: Comparación entre la salida `--log_view` del caso 1/4 y 1/8 con el mismo reflector

```
$ mpiexec -n 2 feenox iaea-3dpwr.fee quarter
geometry = quarter
    keff = 1.02918
   nodes = 70779
    DOFs = 141558
[0/2 tux]   memory = 1.5 Gb (local) 3.0 Gb (global)
[1/2 tux]   memory = 1.5 Gb (local) 3.0 Gb (global)
    wall = 17.0 sec
$ mpiexec -n 4 feenox iaea-3dpwr.fee quarter
geometry = quarter
    keff = 1.02918
   nodes = 70779
    DOFs = 141558
[0/4 tux]   memory = 1.0 Gb (local) 3.9 Gb (global)
[1/4 tux]   memory = 0.9 Gb (local) 3.9 Gb (global)
[2/4 tux]   memory = 1.1 Gb (local) 3.9 Gb (global)
[3/4 tux]   memory = 0.9 Gb (local) 3.9 Gb (global)
    wall = 13.0 sec
$
```

5.3.5. Caso 3D con simetría 1/8, reflector circular resuelto con S_4

Para finalizar el caso, mostramos que FeenoX puede resolver no sólo este problema con el método de difusión sino también con ordenadas discretas. Este archivo de entrada calcula la distribución de flujos que ya hemos mostramos en la figura 1.2 del capítulo 1.

```
PROBLEM neutron_sn 3D GROUPS 2 SN 4 #PROGRESS
READ_MESH iaea-3dpwr-eighth-circular-s4.msh
```



(a) Flujo rápido ϕ_1

(b) Flujo térmico ϕ_2

Figura 5.15.: Flujos del benchmark de 3D PWR de IAEA con simetría 1/8 y reflector circular obtenidos con la formulación de difusión.

```
MATERIAL fuel1     Sigma_s1.2=0.02 Sigma_a1=0.01 Sigma_a2=0.08  nuSigma_f2=0.135
MATERIAL fuel2     Sigma_s1.2=0.02 Sigma_a1=0.01 Sigma_a2=0.085 nuSigma_f2=0.135
MATERIAL fuel2rod  Sigma_s1.2=0.02 Sigma_a1=0.01 Sigma_a2=0.13  nuSigma_f2=0.135
MATERIAL reflector Sigma_s1.2=0.04 Sigma_a1=0    Sigma_a2=0.01  nuSigma_f2=0
MATERIAL reflrod   Sigma_s1.2=0.04 Sigma_a1=0    Sigma_a2=0.055 nuSigma_f2=0

BC vacuum   vacuum
BC mirror   mirror

PRINTF " nodes = %g"   nodes
PRINTF_ALL "solving..."
mumps_icntl_14 = 200
ksp_rtol = 1e-9
penalty_weight=100

SOLVE_PROBLEM
WRITE_RESULTS FORMAT vtk

# print results
PRINTF "  DOFs = %g"   total_dofs
PRINTF "  keff = %.5f" keff
PRINTF "  wall = %.1f sec" wall_time()

mem_global = mpi_memory_global()
mem_avg = mem_global / mpi_size
PRINTF "average memory = %.1f Gb" mem_avg
PRINTF " global memory = %.1f Gb" mem_global
```

```
$ feenox iaea-3dpwr-s4.fee --eps_monitor
 nodes = 3258
[0/1 tux] solving...
  1 EPS nconv=0 first unconverged value (error) 0.996623 (4.34362059e-05)
  2 EPS nconv=0 first unconverged value (error) 0.996626 (4.49580500e-08)
  3 EPS nconv=1 first unconverged value (error) 0.9353 (3.29379374e-08)
  DOFs = 156384
  keff = 0.99663
  wall = 197.9 sec
average memory = 13.6 Gb
 global memory = 13.6 Gb
$ mpiexec -n 4 feenox iaea-3dpwr-s4.fee --eps_monitor
 nodes = 3258
[0/4 tux] solving...
[1/4 tux] solving...
[2/4 tux] solving...
[3/4 tux] solving...
  1 EPS nconv=0 first unconverged value (error) 0.996625 (4.19395044e-05)
  2 EPS nconv=0 first unconverged value (error) 0.996626 (4.67480991e-08)
  3 EPS nconv=1 first unconverged value (error) 0.9353 (3.29361847e-08)
  DOFs = 156384
  keff = 0.99663
  wall = 99.5 sec
average memory = 4.7 Gb
 global memory = 19.0 Gb
$
```

Utilizando el mismo archivo de entrada pero modificando las opciones de la línea de comando es

posible utilizar un solver iterativo para resolver el mismo problema.

```
$ mpiexec -n 12 feenox iaea-3dpwr-s4.fee --eps_monitor --eps_converged_reason --eps_type=jd -- ↩
    st_type=precond  --st_ksp_type=gmres --st_pc_type=asm
 nodes = 3258
[0/12 tux] solving...
[1/12 tux] solving...
[2/12 tux] solving...
[3/12 tux] solving...
[4/12 tux] solving...
[5/12 tux] solving...
[6/12 tux] solving...
[7/12 tux] solving...
[8/12 tux] solving...
[9/12 tux] solving...
[10/12 tux] solving...
[11/12 tux] solving...
  1 EPS nconv=0 first unconverged value (error) 0.542411 (1.45272351e+02)
  2 EPS nconv=0 first unconverged value (error) 0.988282 (6.81535819e+00)
  3 EPS nconv=0 first unconverged value (error) 0.97111 (4.90651112e+00)
  4 EPS nconv=0 first unconverged value (error) 0.991935 (2.72069261e+00)
  5 EPS nconv=0 first unconverged value (error) 0.995226 (1.46652556e+00)
  6 EPS nconv=0 first unconverged value (error) 0.997734 (6.67104794e-01)
  7 EPS nconv=0 first unconverged value (error) 0.996109 (3.82428107e-01)
  8 EPS nconv=0 first unconverged value (error) 0.996941 (2.46293703e-01)
  9 EPS nconv=0 first unconverged value (error) 0.996638 (1.44292090e-01)
 10 EPS nconv=0 first unconverged value (error) 0.996653 (8.93529784e-02)
 11 EPS nconv=0 first unconverged value (error) 0.99657 (8.93529784e-02)
 12 EPS nconv=0 first unconverged value (error) 0.99657 (7.22756164e-02)
 13 EPS nconv=0 first unconverged value (error) 0.996607 (6.32159213e-02)
 14 EPS nconv=0 first unconverged value (error) 0.996704 (4.36139189e-02)
 15 EPS nconv=0 first unconverged value (error) 0.996642 (2.01286172e-02)
 16 EPS nconv=0 first unconverged value (error) 0.996617 (6.63384323e-03)
 17 EPS nconv=0 first unconverged value (error) 0.996629 (2.39149013e-03)
 Linear eigensolve converged (1 eigenpair) due to CONVERGED_TOL; iterations 18
  DOFs = 156384
  keff = 0.99663
  wall = 613.2 sec
average memory = 1.6 Gb
 global memory = 19.4 Gb
```

Observación. La malla es ligeramente más gruesa y de menor orden que en los casos resueltos con difusión en la sección anterior. De todas maneras, la cantidad de grados de libertad es comparable al caso `quarter` de difusión. Pero los recursos computacionales requeridos para resolver un problema de autovalores proveniente de una discretización de ordenadas discretas son significativamente mayores que para difusión.

Observación. Este caso pone en relieve la importancia de la paralelización por MPI: reducir el tiempo de cálculo es un beneficio secundario (que aún debe optimizarse en FeenoX) en comparación con la reducción de la memoria por nodo necesaria para resolver un problema de autovalores con un solver lineal directo. En el caso del solver iterativo, está claro que se intercambia velocidad por memoria.

5.4. El problema de Azmy

> **TL;DR**: Este problema ilustra el "efecto rayo" de la formulación de ordenadas discretas en dos dimensiones. Para estudiar completamente el efecto se necesita rotar la geometría con respecto a las direcciones de S_N.

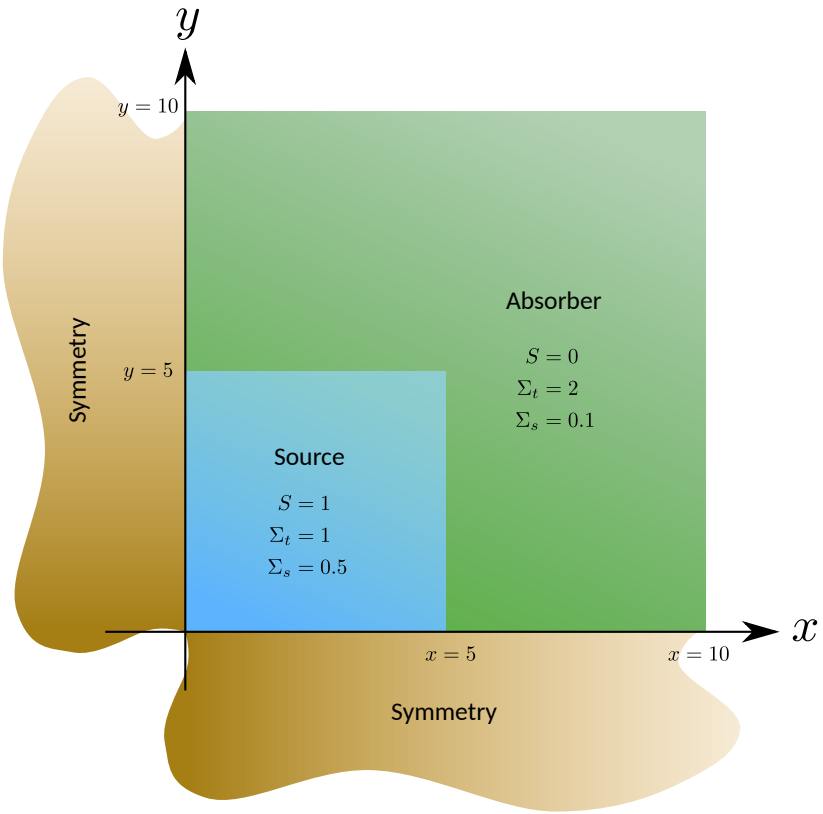


Figura 5.16.: Geometría del problema de Azmy

Este problema adimensional fue introducido en 1988 en el artículo [4] y re-visitado en la tesis de maestría [41]. Consiste en una geometría bi-dimensional muy sencilla, como ilustramos en la figura 5.16. En el paper original, este cuadrado se divide en cuatro cuadrados de 5×5 donde se calculan los flujos medios, que es lo que hacemos en las dos secciones que siguen para comparar los resultados con las referencias.

5.4.1. Malla estructurada uniforme de segundo orden

Aprovechando que la geometría es un cuadrado empezamos resolviendo el problema con una malla estructurada y uniforme de segundo orden (figura 5.17). Mostramos el archivo de entrada de Gmsh para vincular los nombres de las entidades físicas con el archivo de entrada de FeenoX y cómo calculamos los valores medios de los flujos como pide el problema original:

```
// --- geometría ----------------------------
SetFactory("OpenCASCADE");
a = 5;
b = 10;
```

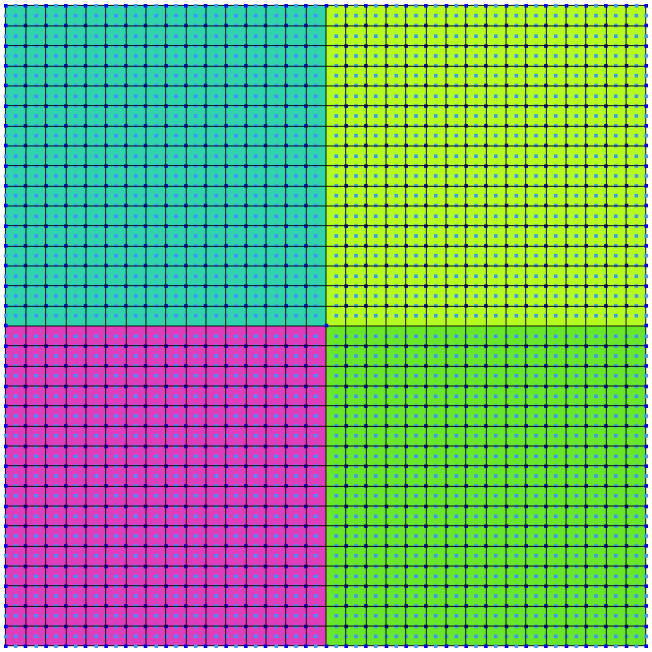


Figura 5.17.: Malla cuadrangular estructurada de segundo orden (quad9) para el problema de Azmy

```
Rectangle(1) = {0, 0, 0, a, a};
Rectangle(2) = {a, a, 0, a, a};
Rectangle(3) = {0, a, 0, a, a};
Rectangle(4) = {a, 0, 0, a, a};
Coherence;

// --- grupos físicos ------------------------
// grupos para materiales
Physical Surface("llq",1) = {1}; // lower left
Physical Surface("lrq",2) = {4}; // lower right
Physical Surface("urq",3) = {2}; // upper right
Physical Surface("ulq",4) = {3}; // upper left

// grupos para condiciones de contornos
Physical Curve("mirror", 13) = {10, 4, 1, 11};
Physical Curve("vacuum", 14) = {9, 7, 6, 12};

Transfinite Curve "*" = 1+16;
Transfinite Surface "*";
Mesh.RecombineAll = 1;
Mesh.ElementOrder = 2;
Mesh.SecondOrderIncomplete = 0;
```

El archivo de entrada de FeenoX es ligeramente más complicado que los anteriores ya que debemos agregar algunas instrucciones para calcular la integral del flujo sobre cada uno de los cuatro cuadrantes con la instrucción `INTEGRATE`:

```
DEFAULT_ARGUMENT_VALUE 1 4
PROBLEM neutron_sn DIM 2 GROUPS 1 SN $1

READ_MESH $0.msh

MATERIAL src S1=1 Sigma_t1=1 Sigma_s1.1=0.5
MATERIAL abs S1=0 Sigma_t1=2 Sigma_s1.1=0.1
```

```
# vinculación entre grupos físicos 2D y materiales
PHYSICAL_GROUP llq MATERIAL src
PHYSICAL_GROUP lrq MATERIAL abs
PHYSICAL_GROUP urq MATERIAL abs
PHYSICAL_GROUP ulq MATERIAL abs

# vinculación entre grupos físicos 1D y condiciones de contorno
BC mirror   mirror
BC vacuum   vacuum

SOLVE_PROBLEM

# calculamos los valores medios en cada cuadrante
INTEGRATE phi1 OVER llq RESULT lower_left_quadrant
INTEGRATE phi1 OVER lrq RESULT lower_right_quadrant
INTEGRATE phi1 OVER urq RESULT upper_right_quadrant

PRINTF "LLQ = %.3e (ref 1.676e+0)" lower_left_quadrant/(5*5)
PRINTF "LRQ = %.3e (ref 4.159e-2)" lower_right_quadrant/(5*5)
PRINTF "URQ = %.3e (ref 1.992e-3)" upper_right_quadrant/(5*5)

WRITE_RESULTS
PRINTF "%g unknowns for S${1}, memory needed = %.1f Gb" total_dofs memory()
```

La ejecución de Gmsh y FeenoX da

```
$ gmsh -2 azmy-structured.geo
[...]
Info    : 4225 nodes 1225 elements
Info    : Writing 'azmy-structured.msh'...
Info    : Done writing 'azmy-structured.msh'
Info    : Stopped on Sat Oct 21 14:30:23 2023 (From start: Wall 0.0222577s, CPU 0.019616s)
$ feenox azmy-structured.fee 2
LLQ = 1.653e+00 (ref 1.676e+0)
LRQ = 4.427e-02 (ref 4.159e-2)
URQ = 2.712e-03 (ref 1.992e-3)
16900 unknowns for S2, memory needed = 0.1 Gb
$ feenox azmy-structured.fee 4
LLQ = 1.676e+00 (ref 1.676e+0)
LRQ = 4.164e-02 (ref 4.159e-2)
URQ = 1.978e-03 (ref 1.992e-3)
50700 unknowns for S4, memory needed = 0.8 Gb
$ feenox azmy-structured.fee 6
LLQ = 1.680e+00 (ref 1.676e+0)
LRQ = 4.120e-02 (ref 4.159e-2)
URQ = 1.874e-03 (ref 1.992e-3)
101400 unknowns for S6, memory needed = 3.3 Gb
$
```

5.4.2. Malla no estructurada localmente refinada de primer orden

Dado que es esperable que haya grandes gradientes en el flujo neutrónico en las interfaces entre la zona de la fuente y el reflector, podemos aprovechar la posibilidad de hacer un refinamiento local

en mallas no estructuradas (figura 5.18). Para ilustrar la flexibilidad de FeenoX, ahora no asignamos una entidad física a cada cuadrante sino que integramos el flujo con el funcional `integrate` dando explícitamente el dominio de integración como función de x e y.

```
// --- geometry -------------------------------------------------
SetFactory("OpenCASCADE");
a = 5;
b = 10;
Rectangle(1) = {0, 0, 0, a, a};
Rectangle(2) = {0, 0, 0, b, b};
Coherence;

// --- physical groups ------------------------------------------
Physical Surface("src") = {1};
Physical Surface("abs") = {2};
Physical Curve("mirror") = {3, 8, 7, 6};
Physical Curve("vacuum") = {4, 5};

// --- meshing options ------------------------------------------
n1 = 8;
n2 = 96;
Mesh.MeshSizeMax = a/n1;
Mesh.MeshSizeMin = a/n2;

Mesh.Algorithm = 6;
Mesh.Optimize = 1;
Mesh.OptimizeNetgen = 1;
Mesh.RecombineAll = 0;
Mesh.ElementOrder = 1;

// local refinements
Field[1] = Distance;
Field[1].CurvesList = {1,2};
Field[1].Sampling = 100;
Field[2] = Threshold;
Field[2].IField = 1;
Field[2].LcMin = Mesh.MeshSizeMin;
Field[2].LcMax = Mesh.MeshSizeMax;
Field[2].DistMin = 4*Mesh.MeshSizeMin;
Field[2].DistMax = 8*Mesh.MeshSizeMin;
Background Field = {2};
```

```
DEFAULT_ARGUMENT_VALUE 1 4
PROBLEM neutron_sn DIM 2 GROUPS 1 SN $1

READ_MESH $0.msh

# podemos dar las secciones eficaces a través de variables
S1_src = 1
Sigma_t1_src = 1
Sigma_s1.1_src = 0.5

S1_abs = 0
Sigma_t1_abs = 2
Sigma_s1.1_abs = 0.1

BC mirror   mirror
BC vacuum   vacuum

SOLVE_PROBLEM
```

```
# calculamos los valores medios con integrales dobles sobre x e y
lower_left_quadrant = integral(integral(phi1(x,y),y,0,5),x,0,5)/(5*5)
lower_right_quadrant = integral(integral(phi1(x,y),y,0,5),x,5,10)/(5*5)
upper_right_quadrant = integral(integral(phi1(x,y),y,5,10),x,5,10)/(5*5)

PRINT %.3e "LLQ" lower_left_quadrant  "(ref 1.676e+0)"
PRINT %.3e "LRQ" lower_right_quadrant "(ref 4.159e-2)"
PRINT %.3e "URQ" upper_right_quadrant "(ref 1.992e-3)"

WRITE_RESULTS
PRINTF "%g unknowns for S${1}, memory needed = %.1f Gb" total_dofs memory()
```

```
$ gmsh -2 azmy.geo
[...]
Info    : 3926 nodes 8049 elements
Info    : Writing 'azmy.msh'...
Info    : Done writing 'azmy.msh'
Info    : Stopped on Sat Oct 21 14:44:46 2023 (From start: Wall 0.129908s, CPU 0.127121s)
$ feenox azmy.fee 2
LLQ     1.653e+00       (ref 1.676e+0)
LRQ     4.427e-02       (ref 4.159e-2)
URQ     2.717e-03       (ref 1.992e-3)
15704 unknowns for S2, memory needed = 0.1 Gb
$ feenox azmy.fee 4
LLQ     1.676e+00       (ref 1.676e+0)
LRQ     4.160e-02       (ref 4.159e-2)
URQ     1.991e-03       (ref 1.992e-3)
47112 unknowns for S4, memory needed = 0.5 Gb
$ feenox azmy.fee 6
LLQ     1.680e+00       (ref 1.676e+0)
LRQ     4.115e-02       (ref 4.159e-2)
URQ     1.890e-03       (ref 1.992e-3)
94224 unknowns for S6, memory needed = 1.9 Gb
$
```

5.4.3. Estudio paramétrico para analizar el “efecto rayo”

En la ref. [41], el autor nota que este tipo de problemas es susceptible al artefacto numérico conocido como “efecto rayo” según el cual la discretización angular hace que algunas direcciones no estén bien representadas por el esquema de S_N. Justamente en esa tesis se proponen formas para lidiar con este efecto. Incluso una de esas formas es discretizar la variable angular con funciones de forma similares a las usadas para discretizar el espacio [38].

Para ilustrar el efecto, se toman perfiles de flujo a lo largo de la dirección y para diferentes valores constantes de x sobre el reflector (donde el flujo es mucho menor que en la fuente como ilustramos en las figuras 5.19 y 5.20. En particular, el autor investiga los siguientes tres valores de x

a. $x = 5.84375$
b. $x = 7.84375$
c. $x = 9.84375$

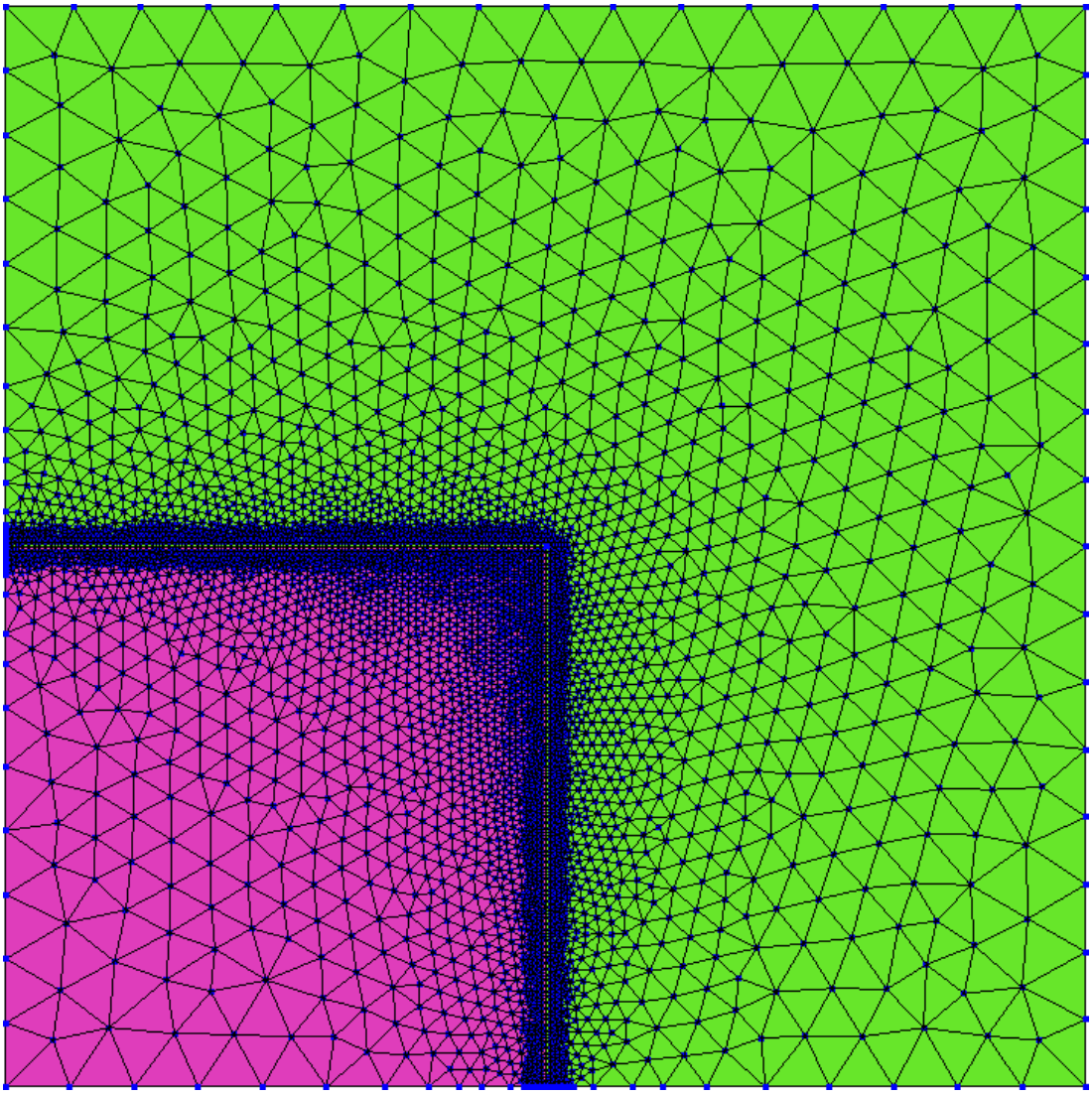
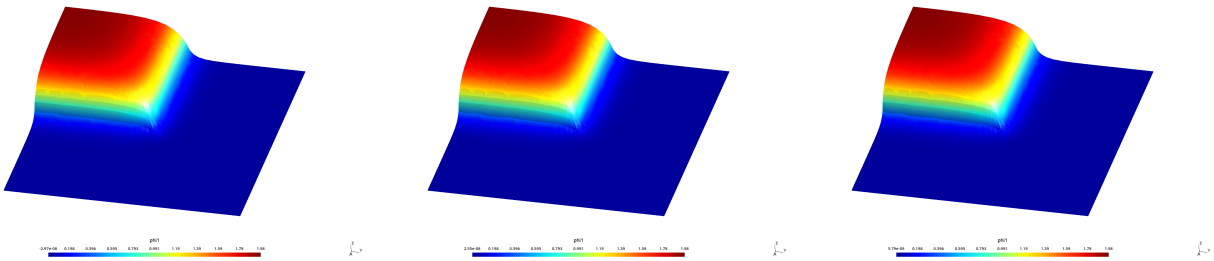
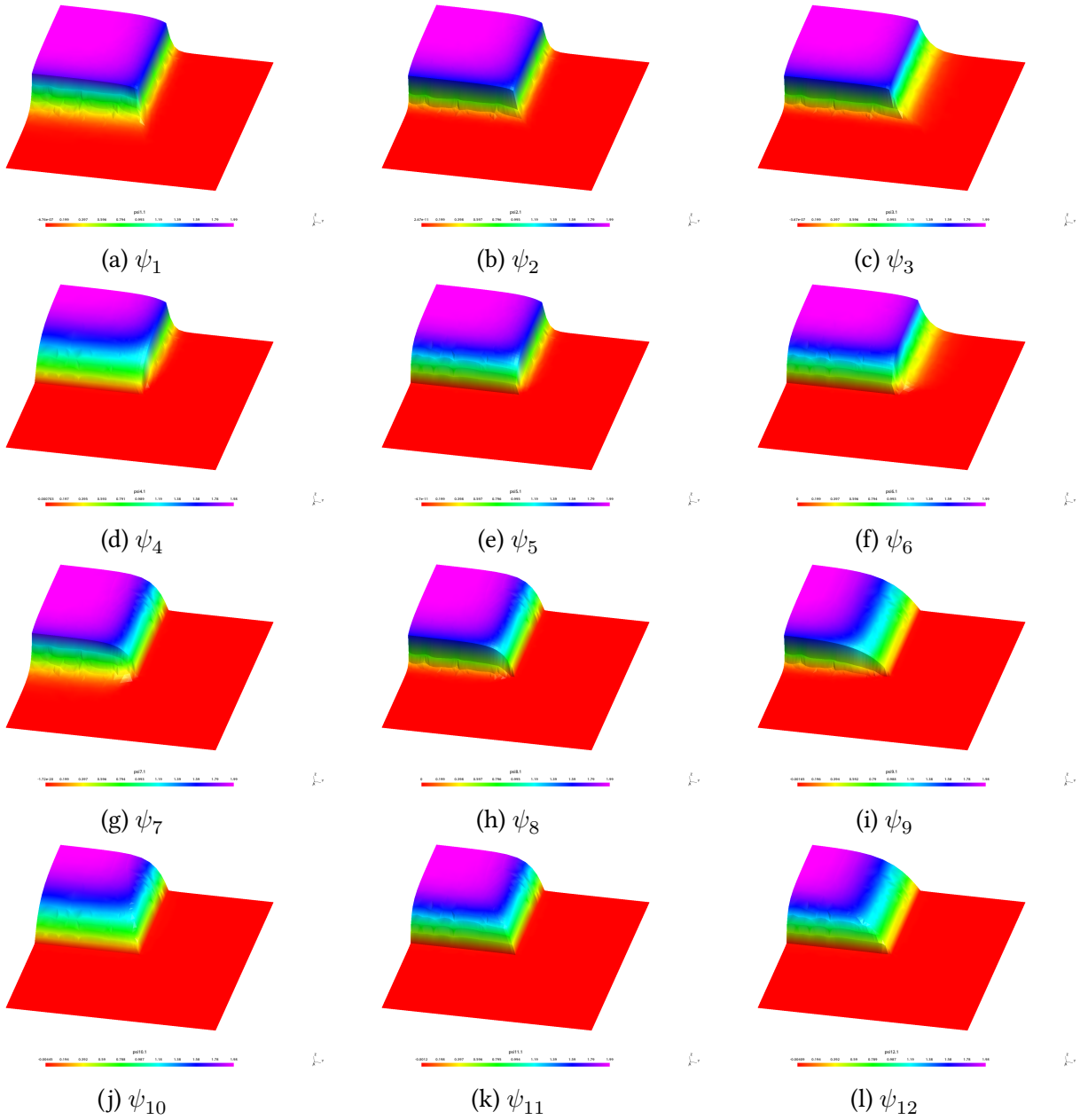


Figura 5.18.: Malla triangular no estructurada de primer orden (tri3) para el problema de Azmy



(a) S_4 (b) S_6 (c) S_8

Figura 5.19.: Flujo escalar $\phi(x, y)$ en el problema de Azmy resuelto con malla no estructurada.



(a) ψ_1 (b) ψ_2 (c) ψ_3

(d) ψ_4 (e) ψ_5 (f) ψ_6

(g) ψ_7 (h) ψ_8 (i) ψ_9

(j) ψ_{10} (k) ψ_{11} (l) ψ_{12}

Figura 5.20.: $1/2 \cdot 4 \cdot (4+2) = 12$ flujos angulares $\psi_g(x, y)$ en el problema de Azmy resuelto con malla no estructurada con S_4.

Para poder entender lo que está pasando, queremos estudiar qué sucede con estos perfiles cuando las direcciones de S_N de alguna manera "rotan" con respecto a la geometría. Como FeenoX usa cuadraturas de nivel simétrico, no podemos rotar las direcciones. Debemos rotar la geometría. Pero si rotamos un ángulo arbitrario θ el cuadrado original no vamos a poder poner las condiciones de simetría ya que se necesita que tanto la dirección incidente como la reflejada estén en el conjunto de cuadraturas. Por lo tanto necesitamos modelar la geometría completa de tamaño 20×20 con condiciones de contorno de vacío en los cuatro lados del cuadrado.

Además del ángulo $\theta \leq 45°$ de rotación de la geometría alrededor del eje z saliendo de la pantalla (o papel si usted está leyendo esta tesis analógicamente), también queremos estudiar qué pasa si variamos la densidad de mallado espacial y angular. Por eso recurrimos a un estudio paramétrico sobre

i. el ángulo θ
ii. un factor c de escala de malla
iii. $N = 4, 6, 8, 10, 12$

Observación. Este caso ilustra la explosión combinatoria de resultados a analizar al realizar estudios paramétricos.

Para eso preparamos un archivo de entrada de Gmsh que

a. toma un argumento `theta` y lo asigna a una variable `angle` para rotar la geometría,
b. define el tamaño de la malla según el factor de escala pasado en la línea de comandos `-clscale`

```
SetFactory("OpenCASCADE");
a = 5;
b = 10;
Rectangle(1) = {-a, -a, 0, 2*a, 2*a};
Rectangle(2) = {a, a, 0, a, a};
Rectangle(3) = {-a, a, 0, 2*a, a};
Rectangle(4) = {a, -a, 0, a, 2*a};
Rectangle(5) = {-2*a, a, 0, a, a};
Rectangle(6) = {-2*a, -a, 0, a, 2*a};
Rectangle(7) = {-2*a, -2*a, 0, a, a};
Rectangle(8) = {-a, -2*a, 0, 2*a, a};
Rectangle(9) = {a, -2*a, 0, a, a};
Coherence;

Rotate {{0, 0, 1}, {0, 0, 0}, angle*Pi/180} {
  Surface{5}; Surface{6}; Surface{7}; Surface{3}; Surface{1}; Surface{8}; Surface{4}; Surface{2};  ↩
      Surface{9};
}

Physical Surface("src", 1) = {1};
Physical Surface("abs", 2) = {4, 2, 3, 5, 6, 7, 8, 9};
Physical Curve("vacuum", 49) = {28, 31, 34, 32, 40, 47, 48, 43, 45, 46, 37, 27};

n = 40/Mesh.MeshSizeFactor;
Transfinite Curve {31, 30, 39, 43, 37, 35, 38, 40} = 1+2*n;
Transfinite Curve {28, 26, 36, 45, 34, 33, 41, 48, 27, 25, 29, 32, 46, 44, 42, 47} = 1+n;
Transfinite Surface "*";
Mesh.RecombineAll = 1;
Mesh.ElementOrder = 1;
Mesh.SecondOrderIncomplete = 0;
```

Por otro lado, el archivo de entrada de FeenoX toma los tres parámetros a estudiar en la línea de comando. Una vez resuelto el problema neutrónico, define tres funciones de una única variable igual a los perfiles de flujo pedidos en la geometría original con $\theta = 0$ y los escribe en un archivo de texto ASCII listos para ser graficados con herramientas como Gnuplot o Pyxplot:

```
DEFAULT_ARGUMENT_VALUE 1 0   # theta
DEFAULT_ARGUMENT_VALUE 2 4   # N
DEFAULT_ARGUMENT_VALUE 3 0   # c
PROBLEM neutron_sn DIM 2 GROUPS 1 SN $2

READ_MESH $0-$1.msh

MATERIAL src S1=1 Sigma_t1=1 Sigma_s1.1=0.5
MATERIAL abs S1=0 Sigma_t1=2 Sigma_s1.1=0.1
BC vacuum   vacuum

SOLVE_PROBLEM

# un poco de trigonometría de prescolar
theta = $1*pi/180
x'(d,x) = d*cos(theta) - x*sin(theta)
y'(d,x) = d*sin(theta) + x*cos(theta)

# perfiles a lo largo de líneas c=cte (en la geometría original)
profile5(x) = phi1(x'(5.84375,x), y'(5.84375,x))
profile7(x) = phi1(x'(7.84375,x), y'(7.84375,x))
profile9(x) = phi1(x'(8.84375,x), y'(9.84375,x))

PRINT_FUNCTION profile5 profile7 profile9 MIN -10 MAX 10 NSTEPS 1000 FILE $0-$1-$2-$3.dat

PRINTF "%g unknowns for S${2} scale factor = ${3}, memory needed = %.1f Gb" total_dofs memory()
```

Finalmente necesitamos un script driver que llame sucesivamente a Gmsh y a FeenoX con las combinaciones de parámetros apropiados. Podemos usar Bash para esto:

```
#!/bin/bash

thetas="0 15 30 45"
cs="4 3 2 1.5 1"
sns="4 6 8 10 12"

for theta in ${thetas}; do
 echo "angle = ${theta};" > azmy-angle-${theta}.geo
 for c in ${cs}; do
  gmsh -v 0 -2 azmy-angle-${theta}.geo azmy-full.geo -clscale ${c} -o azmy-full-${theta}.msh
  for sn in ${sns}; do
   if [ ! -e azmy-full-${theta}-${sn}-${c}.dat ]; then
     echo ${theta} ${c} ${sn}
     feenox azmy-full.fee ${theta} ${sn} ${c} --progress
   fi
  done
 done
done
```

De los muchos archivos con resultados y de las muchas maneras de combinar los perfiles obtenidos, mostramos algunas de las figuras resultantes a continuación. Comencemos con $\theta = 0$ (es decir la geometría original) para $N = 4$, $N = 8$ y $N = 12$ para ver como el perfil "mejora" (figuras 5.21–5.23).

Ahora fijemos c y veamos qué pasa para diferentes ángulos. Algunos valores de θ son "peores" que otros. Parecería que $\theta = 45$ da la "mejor" solución (figuras 5.24–5.27). Para un factor de refinamiento espacial fijo $c = 1$ está claro que aumentar N mejora los perfiles (figuras 5.28–5.30). Finalmente podemos ver cómo cambian los perfiles con el ángulo θ para las mallas más finas (figuras 5.31–5.32).

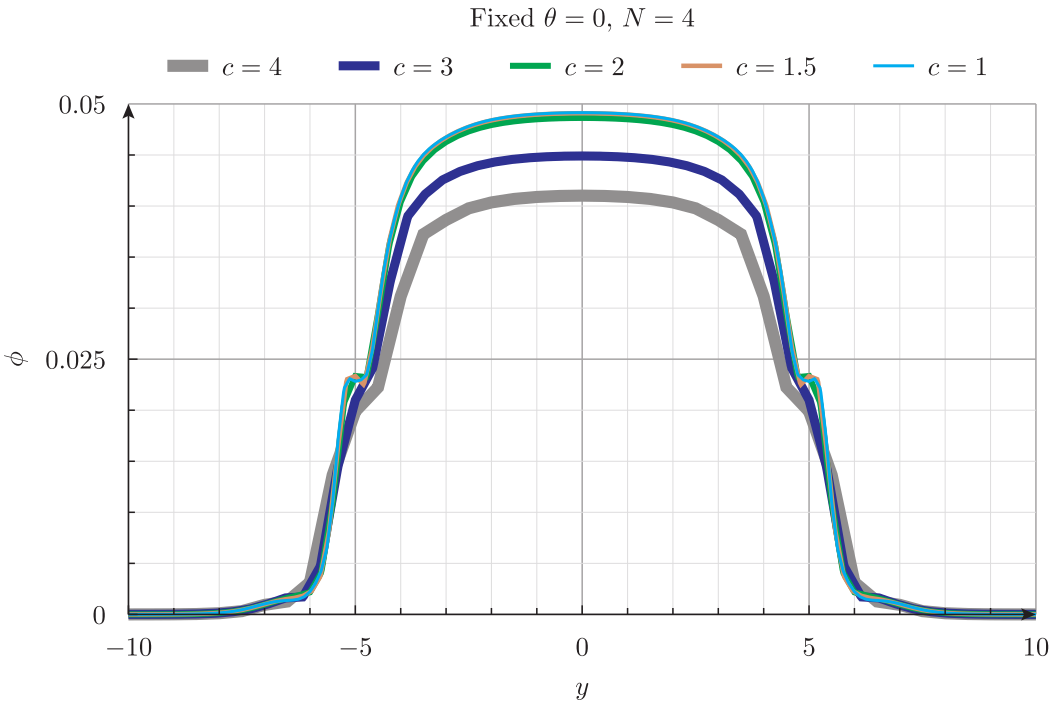


Figura 5.21.: $\theta = 0$ y $N = 4$ para diferentes valores de c

Observación. El análisis detallado del efecto rayo en ordenadas discretas es un posible trabajo futuro derivado de esta tesis de doctorado.

5.5. Benchmarks de criticidad de Los Alamos

> **TL;DR**: Estos problemas proveen una manera de realizar una primera verificación del código con el método de soluciones exactas.

El proceso de verificación de un código numérico involucra justamente, verificar que las ecuaciones se estén resolviendo bien.[7] La forma estricta de realizarlo es comparar alguna medida del error cometido en la solución numérica con respecto a la solución exacta de la ecuación que estamos resolviendo y mostrar que éste tiende a cero con el orden teórico según el método numérico empleado [40]. En particular, la incógnita primaria de una ecuación en derivadas parciales discretizada con el método de elementos finitos debe ir a cero con un order superior en una unidad al orden de los elementos utilizados. Las incógnitas secundarias, con el mismo orden. Es decir, los desplazamientos

[7] Hay un juego de palabras en inglés que indica que verificación quiere decir "are we solving the equations right?" mientras que validación quiere decir "are we solving the right equations?". En la sección 5.9 discutimos más en detalle este concepto.

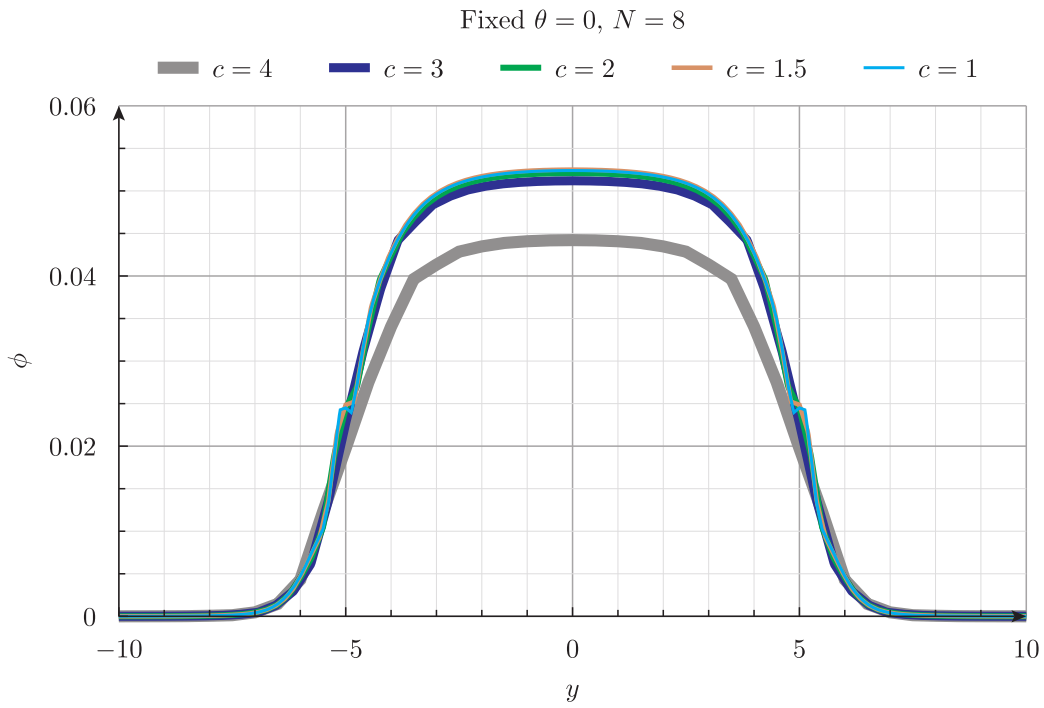


Figura 5.22.: $\theta = 0$ y $N = 8$ para diferentes valores de c

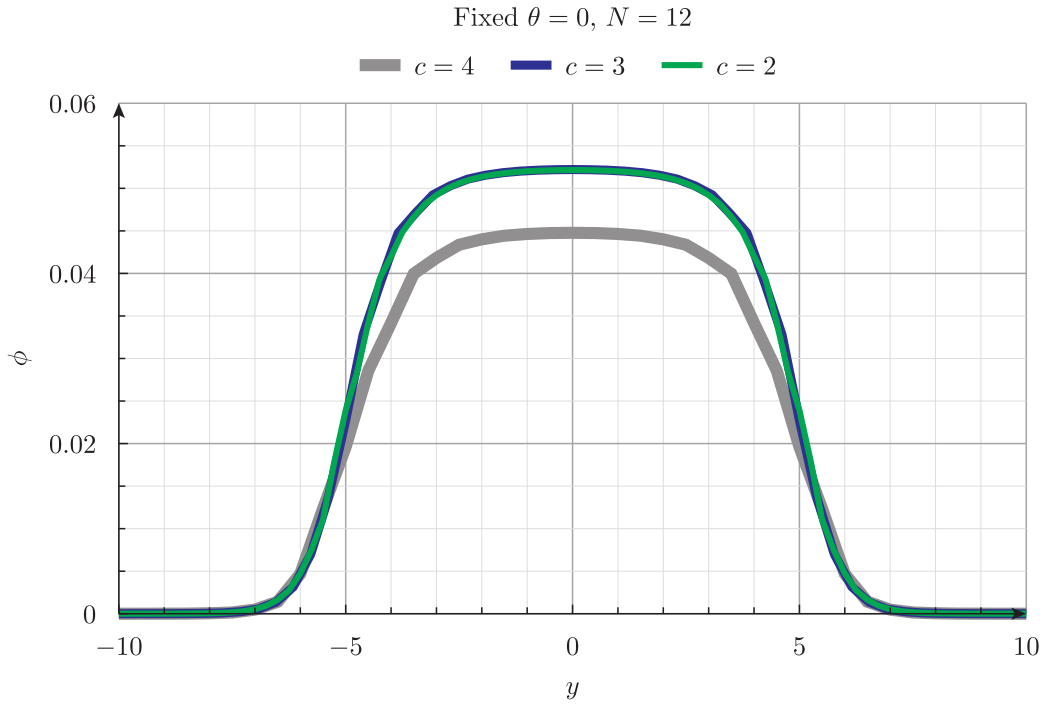


Figura 5.23.: $\theta = 0$ y $N = 12$ para diferentes valores de c

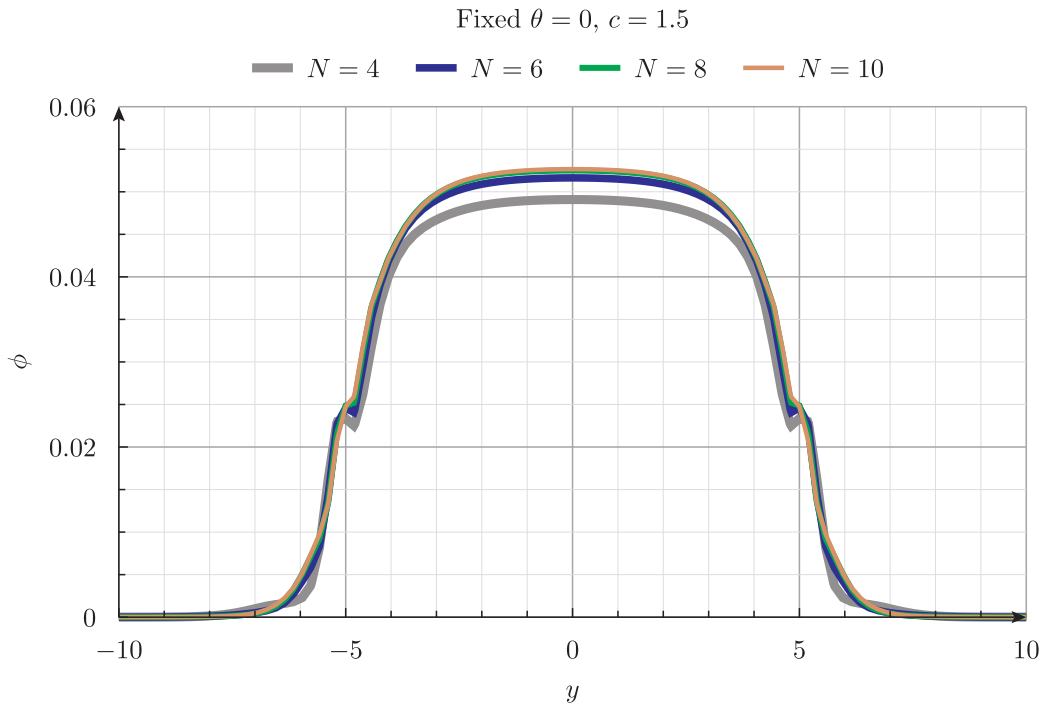


Figura 5.24.: $\theta = 0$ y $c = 1.5$ para diferentes valores de $N

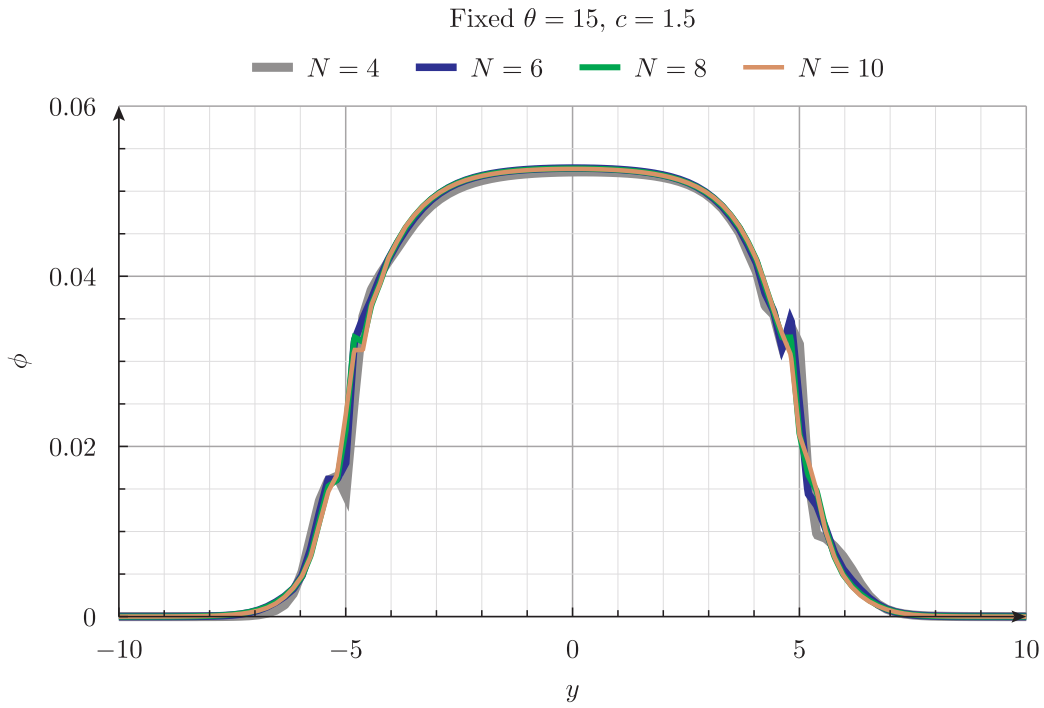


Figura 5.25.: $\theta = 15$ y $c = 1.5$ para diferentes valores de N

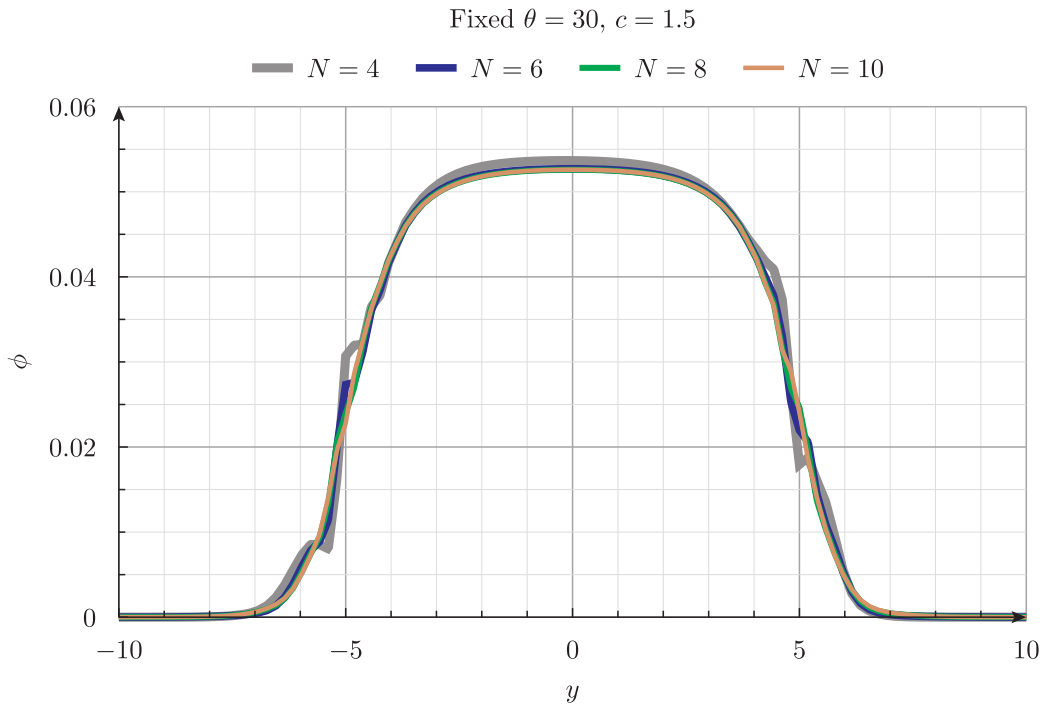


Figura 5.26.: $\theta = 30$ y $c = 1.5$ para diferentes valores de N

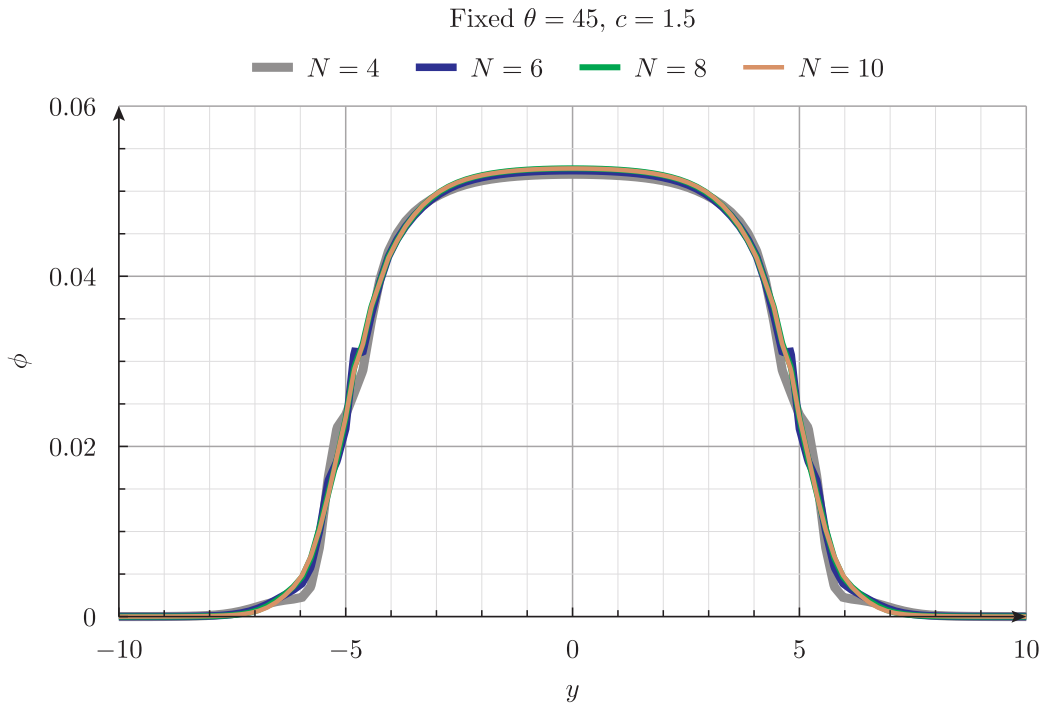


Figura 5.27.: $\theta = 45$ y $c = 1.5$ para diferentes valores de N

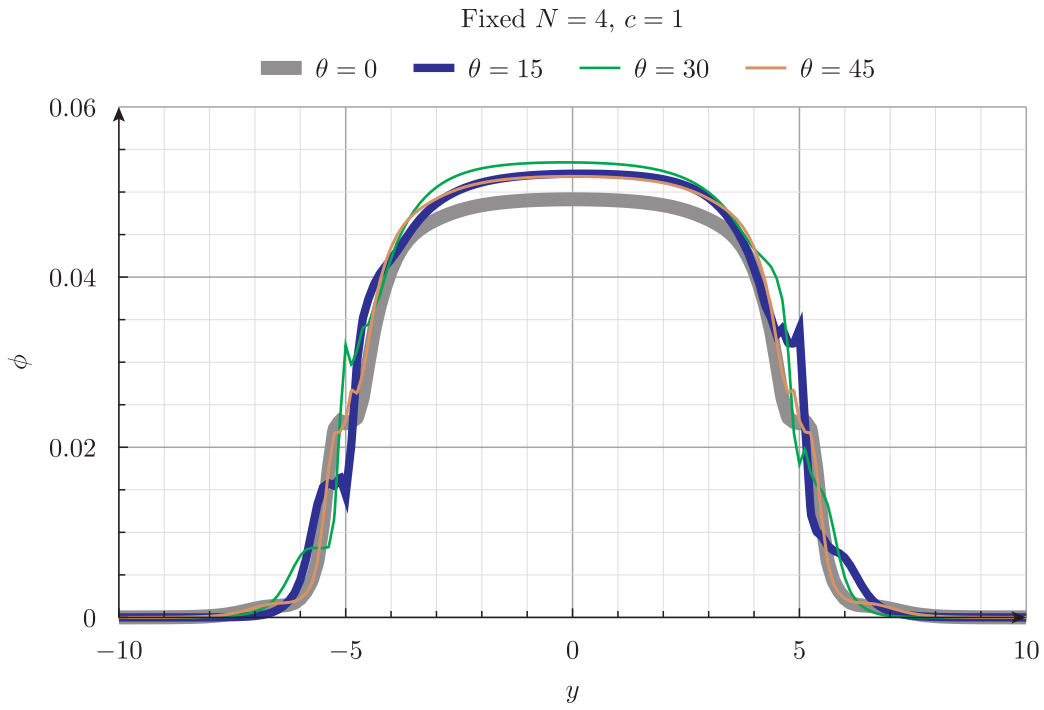


Figura 5.28.: $N = 4$ y $c = 1$ para diferentes valores de θ

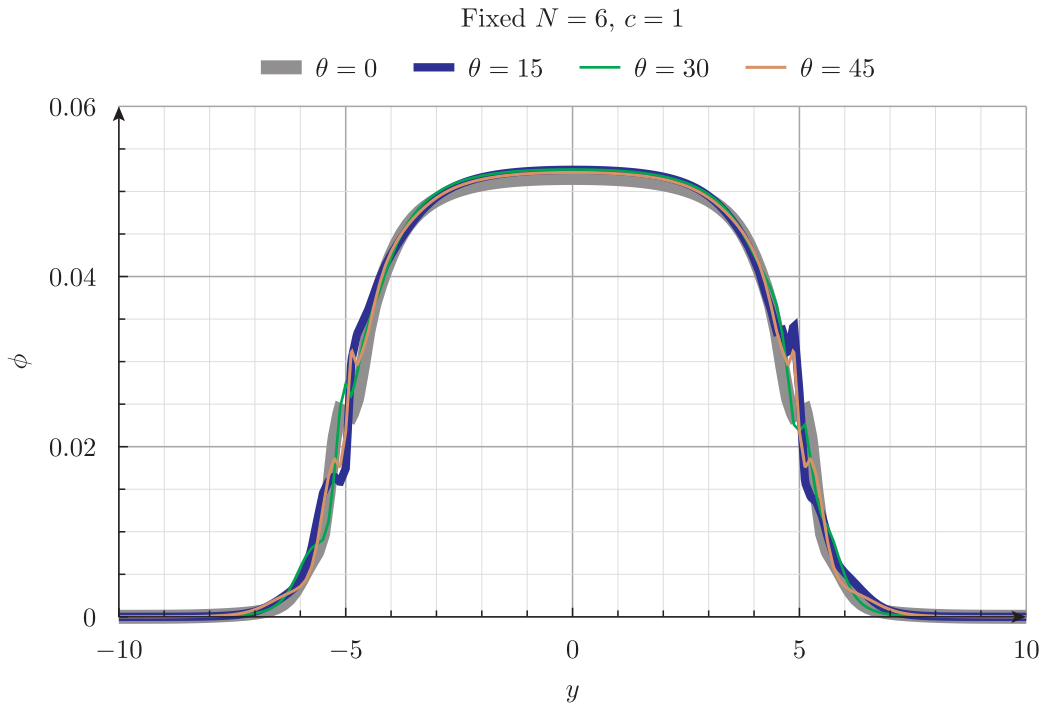


Figura 5.29.: $N = 6$ y $c = 1$ para diferentes valores de θ

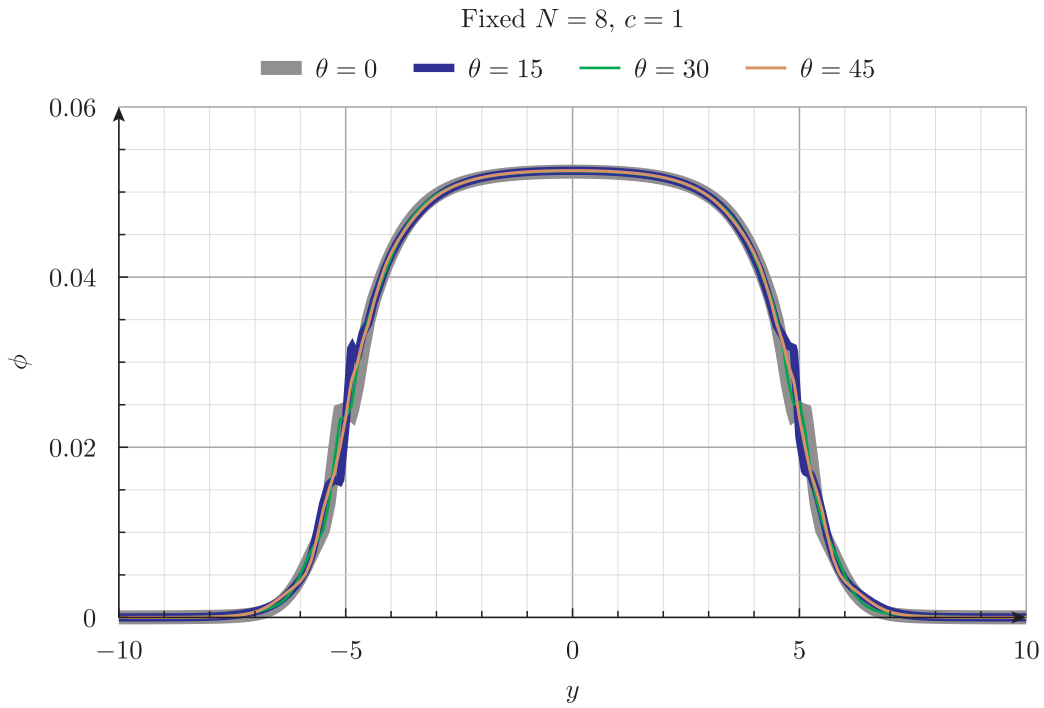


Figura 5.30.: $N = 9$ y $c = 1$ para diferentes valores de θ

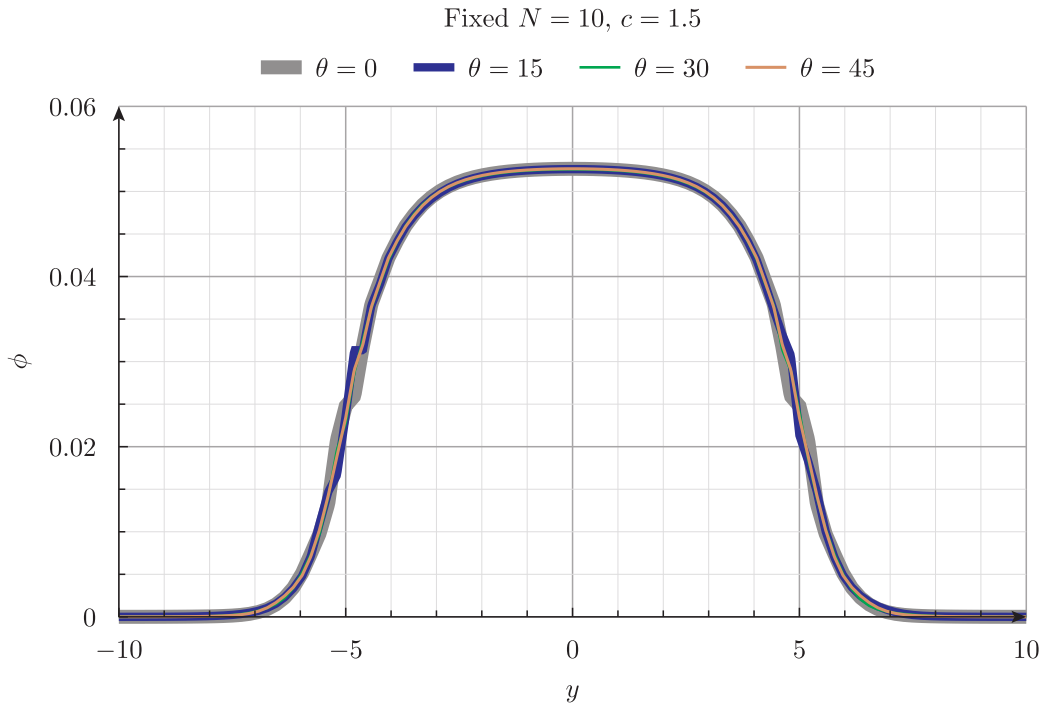


Figura 5.31.: $N = 10$ y $c = 1.5$ para diferentes valores de θ

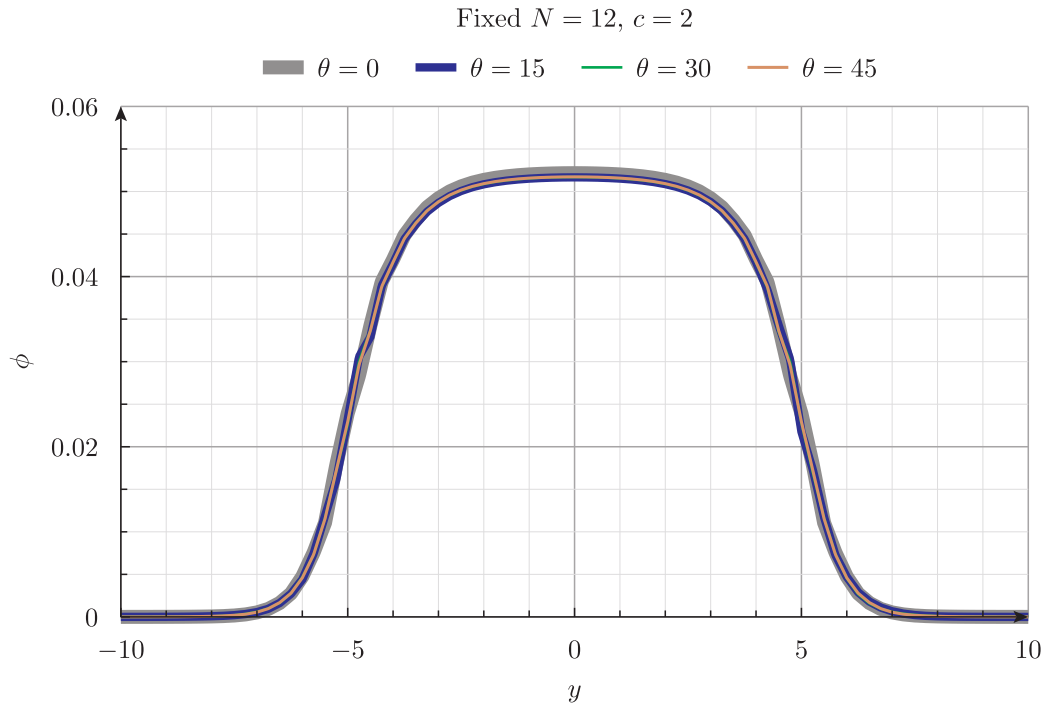


Figura 5.32.: $N = 12$ y $c = 2$ para diferentes valores de θ

en elasticidad y las temperaturas en conducción de calor deben converger a cero como h^3 y las tensiones y los flujos de calor como h^2 si h es el tamaño característico de los elementos de segundo orden utilizados para discretizar el dominio del problema.

Una de las dificultades de la verificación consiste en encontrar la solución de referencia con la cual calcular la medida del error numérico cometido. En la sección 5.9 proponemos una alternativa utilizando el método de soluciones fabricadas,[8] pero en esta sección tomamos algunos de los 75 casos resueltos en la referencia [56] que incluyen

- problemas a uno, dos, tres y seis grupos de energía
- scattering isotrópico y linealmente anisótropo
- geometrías
  - de medio infinito
  - de slab en 1D
  - de círculo en 2D
  - de esfera en 2D
- dominios de uno o varios materiales

El informe provee

a. el factor de multiplicación efectivo infinito k_∞, o
b. el tamaño crítico

[8] Del inglés *Method of Manufactured Solutions*.

de cada una de las 75 configuraciones de reactores en geometrías triviales para diferentes conjuntos de secciones eficaces macroscópicas uniformes a zonas. Cada una de ellas se identifica con una cadena que tiene la forma

Material físil–Material reflector (espesor)—Grupos de energía–Orden de scattering–Geometría

Por ejemplo UD2O-H2O(10)-1-0-SL indica un reactor de uranio y D_2O con un reflector de H_2O de 10 caminos libres medios de espesor, un grupo de energías, scattering isotrópico en geometría tipo slab.

5.5.1. Casos de medio infinito

Una forma de implementar un medio infinito en FeenoX es utilizar una geometría unidimensional tipo slab y poner condiciones de contorno de simetría en ambos extremos. Como el k_∞ no depende ni de la cantidad de nodos espaciales ni de la cantidad de direcciones angulares, reportamos directamente la diferencia entre el k_{eff} calculado por FeenoX con S_2 y el k_∞ de la referencia:

Problema	Identificador	k_∞	$k_\infty - k_{\text{eff}}$
01	PUa-1-0-IN	2.612903	$+2.3 \times 10^{-7}$
05	PUb-1-0-IN	2.290323	$+3.6 \times 10^{-7}$
47	U-2-0-IN	2.216349	-4.6×10^{-7}
50	UAl-2-0-IN	2.661745	-4.2×10^{-7}
70	URRa-2-1-IN	1.631452	-2.5×10^{-6}
74	URR-3-0-IN	1.60	$+2.7 \times 10^{-15}$
75	URR6-6-0-IN	1.60	$+5.2 \times 10^{-14}$

Tabla 5.7.: Factores de multiplicación de medio infinito con las cifras significativas reportadas en [56] y diferencia con respecto al k_{eff} calculado con FeenoX con S_2.

Observación. Está claro que la diferencia $k_\infty - k_{\text{eff}}$ es menor a la precisión del k_∞ dado en la referencia y que los problemas 74 y 75 han sido "manufacturados" para que el factor infinito sea exactamente igual a 8/5.

Para ilustrar cómo hemos obtenido la tabla 5.7 mostramos a continuación el archivo de entrada del caso 74:

```
# Los Alamos LA-13511 Analytical Benchmark Test Set for Criticallity Code Verification
# problem 74
PROBLEM neutron_sn 1D GROUPS 3 SN 2
READ_MESH la-IN.msh
INCLUDE URR.fee
BC mirror  mirror
penalty_weight = 10
SOLVE_PROBLEM
PRINT %+.1e keff-1.60
```

Table 59: Fast Energy Group Macroscopic Cross Sections (cm^{-1}) for Research Reactor

Material	ν_3	Σ_{3f}	Σ_{3c}	Σ_{33s}	Σ_{23s}	Σ_{13s}	Σ_3	χ_3
Research Reactor	3.0	0.006	0.006	0.024	0.171	0.033	0.240	0.96

Table 60: Middle Energy Group Macroscopic Cross Sections (cm^{-1}) for Research Reactor

Material	ν_2	Σ_{2f}	Σ_{2c}	Σ_{22s}	Σ_{32s}	Σ_{12s}	Σ_2	χ_2
Research Reactor	2.50	0.060	0.040	0.60	0.0	0.275	0.975	0.04

Table 61: Slow Energy Group Macroscopic Cross Sections (cm^{-1}) for Research Reactor

Material	ν_1	Σ_{1f}	Σ_{1c}	Σ_{11s}	Σ_{21s}	Σ_{31s}	Σ_1	χ_1
Research Reactor	2.0	0.90	0.20	2.0	0.0	0.0	3.10	0.0

Figura 5.33.: Secciones eficaces macroscópicas a tres grupos del material URR [56].

Como las secciones eficaces son las mismas para varios problemas, cada material tiene un archivo separado que se incluye desde cada entrada principal. En este caso, a partir de los datos originales mostrados en la figura 5.33, preparamos el archivo `URR.fee`:

```
# material Research Reactor (tables 59, 60 and 61)
# NOTE: 1 is the slow energy grop and 3 is the fast energy group
# fast-energy group
nu3_fuel = 3.0
Sigma_f3_fuel = 0.006
nuSigma_f3_fuel = nu3_fuel*Sigma_f3_fuel

Sigma_s3.3_fuel     = 0.024
Sigma_s3.2_fuel     = 0.171
Sigma_s3.1_fuel     = 0.033

Sigma_t3_fuel   = 0.240
chi[3] = 0.96

# middle-energy group
nu2_fuel = 2.5
Sigma_f2_fuel = 0.060
nuSigma_f2_fuel = nu2_fuel*Sigma_f2_fuel

Sigma_s2.3_fuel     = 0
Sigma_s2.2_fuel     = 0.60
Sigma_s2.1_fuel     = 0.275

Sigma_t2_fuel   = 0.975
chi[2] = 0.04

# slow energy group
nu1_fuel = 2.0
Sigma_f1_fuel = 0.90
nuSigma_f1_fuel = nu1_fuel*Sigma_f1_fuel

Sigma_s1.3_fuel     = 0
Sigma_s1.2_fuel     = 0
Sigma_s1.1_fuel     = 2.0

Sigma_t1_fuel   = 3.10
chi[1] = 0.0
```

Observación. La definición de los grupos en la referencia [56] es opuesta a la de la mayoría de la literatura: el grupo uno es el térmico y el de mayor energía es el de mayor índice g.

5.5.2. Casos de medio finito

La forma de "verificar" que FeenoX resuelve razonablemente bien problemas de transporte de neutrones con dependencia espacial es entonces

1. crear una malla con el tamaño indicado como crítico en cada problema, y
2. mostrar que el factor de multiplicación efectivo k_{eff} obtenido se acerca a la unidad a medida que aumentamos el tamaño del problema discretizado, sea por refinar la malla espacial o por aumentar el número N de ordenadas discretas.

Para ello podemos crear un script de Bash que llame a cada uno de los archivos de entrada de cada problema, luego de haber creado mallas con diferente refinamiento, con valores de N sucesivos:

```
#!/bin/bash

for p in $(ls la-*.fee | grep -v IN); do
 rm -f tmp
 for c in 1 0.75 0.65 0.60 0.55 0.5; do
  gmsh -v 0 -3 $(basename ${p} .fee).geo -clscale ${c}
  for N in 2 4 6 8; do
   feenox ${p} $N | tee -a tmp
  done
 done
 sort -g tmp > $(basename ${p} .fee).csv
done
```

La figura 5.34 muestra—en forma poco rigurosa—que en general al aumentar el tamaño del problema resuelto por FeenoX, el factor de multiplicación efectivo se acerca a la unidad. Esta no es una verificación según la definición industrial de "code verification" pero indica que nuestro solver hace las cosas razonablemente bien, incluso en casos con scattering anisótropo y con más de dos grupos de energías.

5.6. Slab a dos zonas: efecto cúspide por dilución de XSs

> **TL;DR**: Este problema ilustra el error cometido al analizar casos multi-material con mallas estructuradas donde la interfaz no coincide con los nodos de la malla (y la flexibilidad de FeenoX para calcular y comparar soluciones analíticas con soluciones numéricas).

Richard Stallman dice en sus conferencias (incluso en castellano) "la mejor manera de resolver un problema es evitar tenerlo". Como discutimos en la sección 1.2, los códigos neutrónicos de núcleo que usamos durante el completamiento de la Central Nuclear Atucha II solamente usaban mallas estructuradas. Además del inconveniente que esto supone para modelar barras de control invertidas, muy a menudo teníamos que lidiar con un efecto numérico denominado "cúspide por dilución de secciones eficaces". Este efecto aparece cuando la posición de una barra de control no coincide con

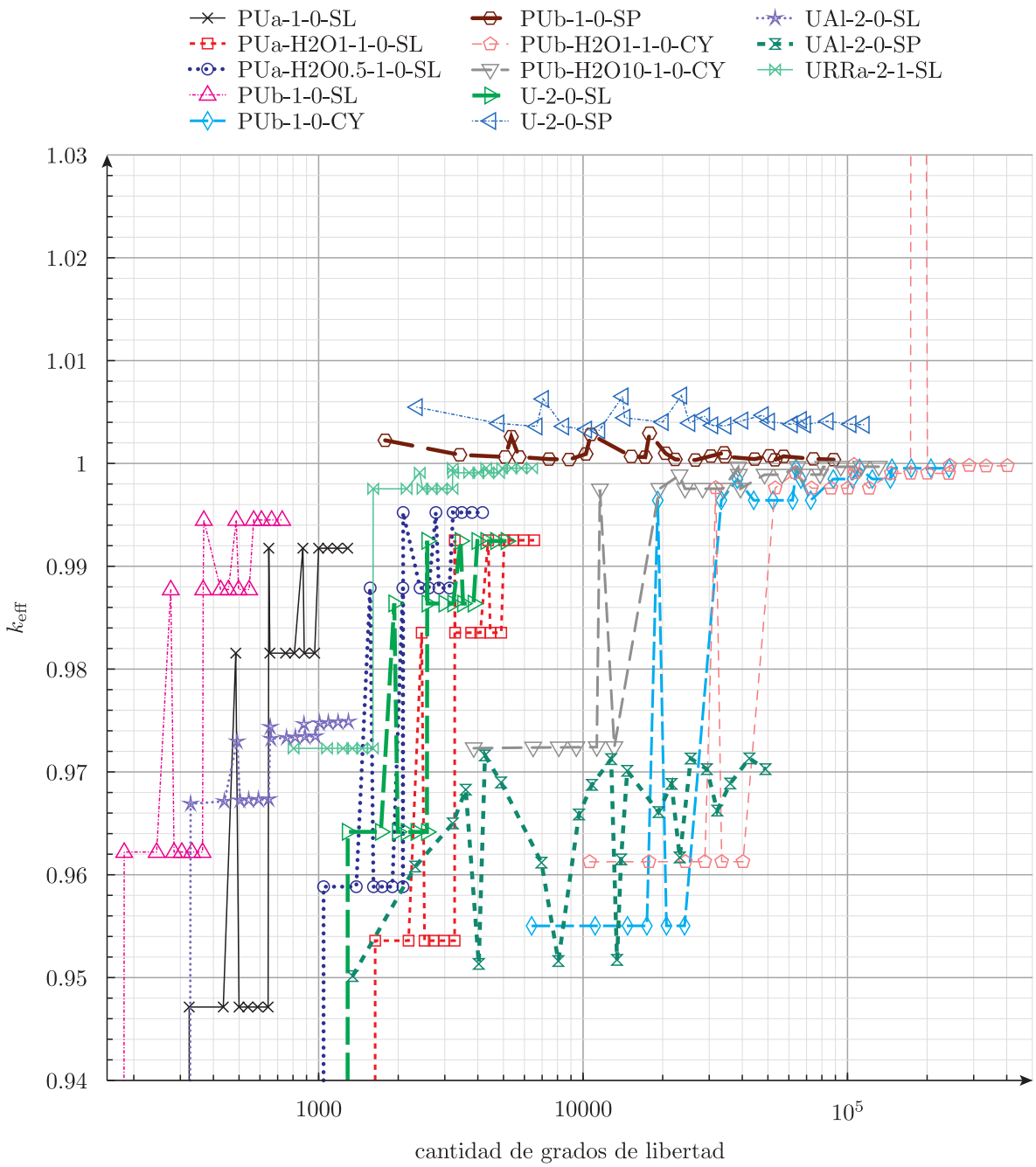


Figura 5.34.: Factor de multiplicación vs. cantidad de grados de libertad para 13 de los 75 problemas de [56]. A medida que aumentamos el tamaño del problema (sea por refinamiento de malla o por incrementar N) el k_{eff} se acerca a la unidad.

la interfaz entre dos celdas de cálculo y hay que de alguna manera "diluir" las secciones eficaces de la barra absorbente entre las secciones eficaces del tubo guía vacío en forma proporcional a la posición geométrica de la barra en la celda de cálculo.

Es exactamente este efecto el que ilustramos en este ejemplo, pero en geometría tipo slab ya que dicho problema tiene solución analítica exacta (en difusión, que es lo que usaban los códigos en Atucha de cualquier manera). Efectivamente, consideremos un reactor en geometría slab a dos zonas, como ilustramos en la figura 5.35:

a. La zona A tiene $k_\infty < 1$ y ocupa el intervalo $0 < x = a$, y
b. La zona B tiene $k_\infty > 1$ y ocupa el intervalo $a < x < b$.

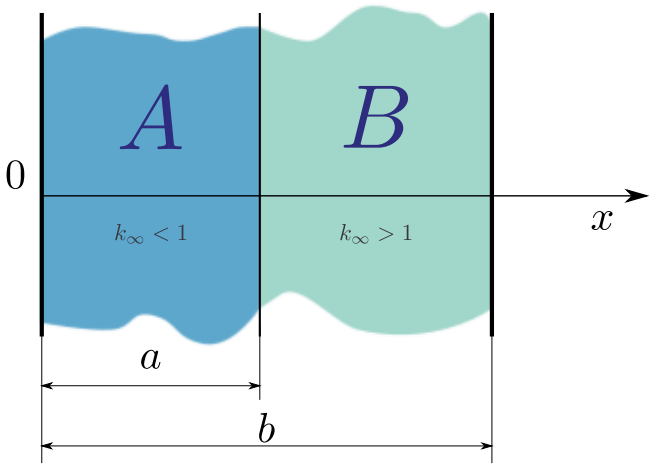


Figura 5.35.: Slab a dos zonas, una A con $k_\infty < 1$ y otra B con $k_\infty > 1$

Si

1. resolvemos el slab con un grupo de energías con aproximación de difusión,
2. las dos zonas tienen secciones eficaces macroscópicas uniformes, y
3. hacemos que el flujo escalar ϕ sea cero en ambos extremos $x = 0$ y $x = b$,

entonces el factor efectivo de multiplicación k_{eff} es tal que

$$\begin{aligned} &\sqrt{D_A \cdot \left(\Sigma_{aA} - \frac{\nu\Sigma_{fA}}{k_{\text{eff}}}\right)} \cdot \tan\left[\sqrt{\frac{1}{D_B} \cdot \left(\frac{\nu\Sigma_{fB}}{k_{\text{eff}}} - \Sigma_{aB}\right)} \cdot (a-b)\right] \\ &-\sqrt{D_B \cdot \left(\frac{\nu\Sigma_{fB}}{k_{\text{eff}}} - \Sigma_{aB}\right)} \cdot \tanh\left[\sqrt{\frac{1}{D_A} \cdot \left(\Sigma_{aA} - \frac{\nu\Sigma_{fA}}{k_{\text{eff}}}\right)} \cdot b\right] = 0 \end{aligned} \tag{5.1}$$

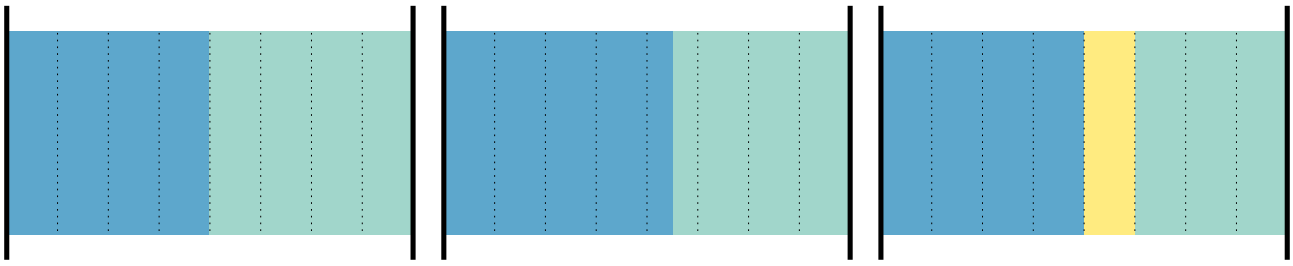
Observación. Aunque no lo parezca, esta ecuación 5.1 *es* la solución analítica para k_{eff}. Lo que hay que hacer para obtener su valor es resolver esta ecuación implícita, cosa que FeenoX puede hacer perfectamente como mostramos a continuación.

Observación. Este problema no sólo tiene solución analítica para el factor k_{eff} sino que también el flujo $\phi(x)$ tiene una expresión algebraica por trozos para $0 < x < a$ y para $a < x < b$. De hecho una para $x < a$ y otra para $x > a$. Dicha solución no es relevante para este problema, pero

por completitud dejamos comentadas las instrucciones de FeenoX para evaluarlo y escribirlo en un archivo de salida.

Por otro lado, vamos a calcular k_{eff} numéricamente de dos maneras diferentes, a saber:

i. usando una malla no uniforme con n elementos y $n+1$ nodos de forma tal que siempre haya un nodo *exactamente* en la interfaz $x = a$ para cualquier valor arbitrario de b, y
ii. con una malla uniforme con n elementos de igual tamaño y $n+1$ nodos equi-espaciados para emular el comportamiento de los solvers que no pueden manejar el caso i. Si la interfaz coincide exactamente con uno de los nodos, entonces hay dos zonas bien definidas (figura 5.36a). Pero en general, esto no va a suceder (figura 5.36b). Entonces, al elemento que contiene la interfaz $x = a$ le asignamos un pseudo material AB (figura 5.36c) cuyas secciones eficaces son un promedio pesado de las de A y B según la fracción geométrica que cada una de las zonas ocupa en el elemento. Es decir, si $b = 100$ y $n = 10$ entonces cada elemento tiene un ancho igual a 10. Si además $a = 52$ entonces este material AB tendrá un 20% del material A y un 80% del material B.



(a) a coincide con un nodo (b) a no coincide con un nodo (c) Pseudo-material AB

Figura 5.36.: Manejo de la interfaz en el caso ii de malla estructurada.

Para ello, preparamos dos archivos de Gmsh. Primero `two-zone-slab-i.geo` para el caso no uniforme (que es más sencillo para Gmsh que el caso uniforme):

```
Merge "ab.geo";

Point(1) = {0, 0, 0, lc+lc/n};
Point(2) = {a, 0, 0, lc+lc/n};
Point(3) = {b, 0, 0, lc+lc/n};

Line(1) = {1, 2};
Line(2) = {2, 3};
Physical Line("A", 1) = {1};
Physical Line("B", 2) = {2};

Physical Point("left") = {1};
Physical Point("right") = {3};
```

y luego el archivo `two-zone-slab-ii.geo`:

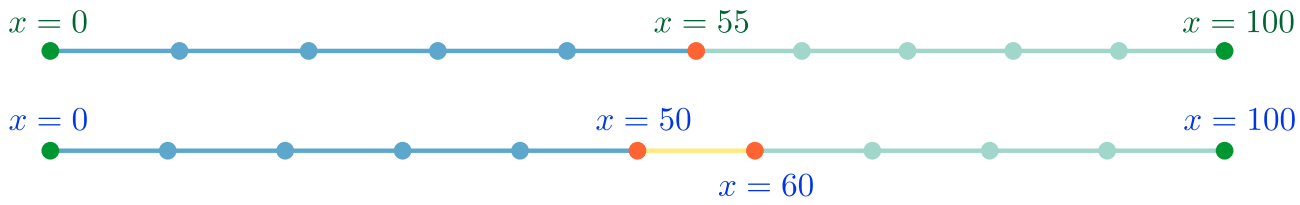
```
Merge "ab.geo";
a_left  = Floor(a/lc)*lc;
a_right = (Floor(a/lc)+1)*lc;

Point(1) = {0, 0, 0, lc};
Point(2) = {a_left, 0, 0, lc};
```

```
Point(3) = {a_right, 0, 0, lc};
Point(4) = {b, 0, 0, lc};

Line(1) = {1, 2};
Line(2) = {2, 3};
Line(3) = {3, 4};
Physical Line("A", 1) = {1};
Physical Line("AB", 3) = {2};
Physical Line("B", 2) = {3};

Physical Point("left") = {1};
Physical Point("right") = {4};
```

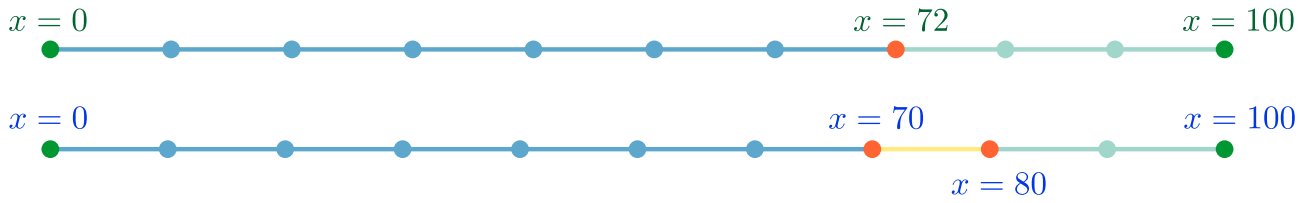
Ambos necesitan un archivo `ab.geo` con la definición de las variables geométricas necesarias:

```
// dimensiones del slab
a = 57;
b = 100;
// número de elementos
n = 20;
// longitud característica
lc = b/n;
```

La figura 5.37 muestra la diferencia entre los dos casos i (no uniforme) y ii (uniforme) para $n = 10$. Cuando $a = 55$ (figura 5.37a), en la malla i hay cinco elementos en cada una de las dos zonas. Los elementos de la zona A son ligeramente más grandes que los de la zona B. En la malla ii todos los elementos son iguales. Hay cinco elementos en la zona A, uno en la zona AB y cuatro en la zona B. Para $a = 72$ (figura 5.37b), hay siete elementos a la izquierda de $x = a$ y tres a la derecha en el caso i. En el caso ii, hay siete elementos en la zona A, uno en la zona AB y dos en la zona B.



(a) $a = 55$



(b) $a = 72$

Figura 5.37.: Mallas i (arriba, no uniforme con dos materiales) y ii (abajo, uniforme con tres materiales) para $n = 10$.

Ahora preparamos este archivo de entrada de FeenoX que es de los más complicados (pero a la vez de los más flexibles) que hemos visto hasta el momento:

```
PROBLEM neutron_diffusion 1D
DEFAULT_ARGUMENT_VALUE 1 i

# leemos la malla según el argumento sea i o ii
READ_MESH two-zone-slab-$1.msh

# este archivo ab.geo tiene información geométrica y es creado
# por el script en bash que nos llama
INCLUDE ab.geo

# XSs para el material puro A de x=0 to x=a
D1_A = 0.5
Sigma_a1_A = 0.014
nuSigma_f1_A = 0.010

# XSs para el material puro B de x=a to x=b
D1_B = 1.2
Sigma_a1_B = 0.010
nuSigma_f1_B = 0.014

# psedo-material AB usado solamente si $1 es uniform
a_left  = floor(a/lc)*lc;
xi = (a - a_left)/lc
Sigma_tr_A = 1/(3*D1_A)
Sigma_tr_B = 1/(3*D1_B)
Sigma_tr_AB = xi*Sigma_tr_A + (1-xi)*Sigma_tr_B
D1_AB = 1/(3*Sigma_tr_AB)
Sigma_a1_AB = xi * Sigma_a1_A + (1-xi)*Sigma_a1_B
nuSigma_f1_AB = xi * nuSigma_f1_A + (1-xi)*nuSigma_f1_B

# condiciones de contorno de flujo nulo
BC left  phi1=0
BC right phi1=0

SOLVE_PROBLEM

# calculamos el keff analítico como la raíz de F1(k)-F2(k)
F1(k) = sqrt(D1_A*(Sigma_a1_A-nuSigma_f1_A/k)) *  ↩
    tan(sqrt((1/D1_B)*(nuSigma_f1_B/k-Sigma_a1_B))*(a-b))
F2(k) = sqrt(D1_B*(nuSigma_f1_B/k-Sigma_a1_B)) *  ↩
    tanh(sqrt((1/D1_A)*(Sigma_a1_A-nuSigma_f1_A/k))*b)
k = root(F1(k)-F2(k), k, 1, 1.2)

# el flujo analítico en función de x (no es usado en este problema)
B_A = sqrt((Sigma_a1_A - nuSigma_f1_A/k)/D1_A)
fluxA(x) = sinh(B_A*x)

B_B = sqrt((nuSigma_f1_B/k - Sigma_a1_B)/D1_B)
fluxB(x)= sinh(B_A*b)/sin(B_B*(a-b)) * sin(B_B*(a-x))

# factor de normalización
f = a/(integral(fluxA(x), x, 0, b) + integral(fluxB(x), x, b, a))
flux(x) := f * if(x < b, fluxA(x), fluxB(x))

# esribimos keff (numerico) y k (analitico) vs. a
PRINT a keff k keff-k

# si quisiéramos, con estas líneas podríamos comparar los flujos
# PRINT_FUNCTION flux MIN 0 MAX a STEP a/1000 FILE_PATH two-zone-analytical.dat
# PRINT_FUNCTION phi1 phi1(x)-flux(x)         FILE_PATH two-zone-numerical.dat
```

La ejecución necesita un argumento que puede ser `i` o `ii` según sea el punto que queremos resolver:

i. malla no uniforme
ii. malla uniforme

Cada uno de los dos puntos usa una malla diferente, que luego explicamos cómo generamos. Luego incluimos el mismo archivo `ab.geo` con la información geométrica que ya incluyeron los archivos de entradas de Gmsh. Como la sintaxis de asignación de variables es igual en Gmsh que en Feenox, podemos incluirlo directamente (el punto y coma al finalizar la línea es opcional en FeenoX).

A continuación definimos las secciones eficaces de los dos materiales asignando variables y finalmente calculamos las secciones eficaces del pseudo-material AB utilizando las facilidades algebraicas que nos provee FeenoX. Ponemos condiciones de contorno nulas, resolvemos el problema y tenemos en la variable especial `keff` el factor de multiplicación efectiva para una posición `a` de la interfaz en un slab de ancho `b` en el caso `i` (no uniforme) o `ii` (uniforme).

Lo que queremos ahora es comparar este `keff` numérico con el k_{eff} analítico para cuantificar los errores producidos por los métodos `i` y `ii`. Procedemos entonces a resolver (numéricamente) la ecuación 5.1 con el funcional `root()` y almacenamos el resultado en la variable `k`, que luego podemos imprimir en la salida estándar y redireccionar a un archivo de texto para graficar estas diferencias.

Observación. Hemos necesitado resolver numéricamente la ecuación 5.1 para obtener el k_{eff}, pero éste sigue siendo un valor analítico. La búsqueda numérica de la raíz de la ecuación es un algoritmo que puede continuar hasta que el error cometido sea arbitrariamente pequeño, por lo que `k` es *analítico* y `keff` *numérico*.

Nos falta un archivo más para completar el estudio, que es un script que haga el barrido de a en un cierto intervalo, y llame a Gmsh y a FeenoX para los casos *i* y *ii*:

```
#!/bin/bash

b="100"   # ancho adimensional del slab
if [ -z $1 ]; then
  n="10"    # número de elementos
else
  n=$1
fi

rm -rf two-zone-slab-*-${n}.dat

# barremos a (posición de la interfaz)
for a in $(seq 35 57); do
  cat << EOF > ab.geo
a = ${a};
b = ${b};
n = ${n};
lc = b/n;
EOF
  for p in i ii; do
    gmsh -1 -v 0 two-zone-slab-${p}.geo
    feenox two-zone-slab.fee ${p} | tee -a two-zone-slab-${p}-${n}.dat
  done
done
```

Estamos entonces en condiciones de ejecutar este script para poder graficar los errores de ambos métodos:

```
$ ./two-zone-slab.sh 10
35      1.19332 1.19768 -0.00435617
35      1.18588 1.19768 -0.0117998
36      1.18882 1.19333 -0.0045144
[...]
56      1.0477  1.06819 -0.0204869
57      1.05031 1.05931 -0.00899971
57      1.04029 1.05931 -0.0190241
$ ./two-zone-slab.sh 20
35      1.19647 1.19768 -0.00120766
35      1.19647 1.19768 -0.00120766
36      1.19208 1.19333 -0.00125124
[...]
56      1.06328 1.06819 -0.00490434
57      1.05725 1.05931 -0.00206549
57      1.05303 1.05931 -0.00628299
$
```

La figura 5.38 ilustra cabalmente el punto de Richard Stallman: en lugar de lidiar con cómo corregir el efecto "cúspide" (por ejemplo modificando la posición de la barra de control artificialmente para reducirlo) es mucho más efectivo evitarlo en primer lugar.

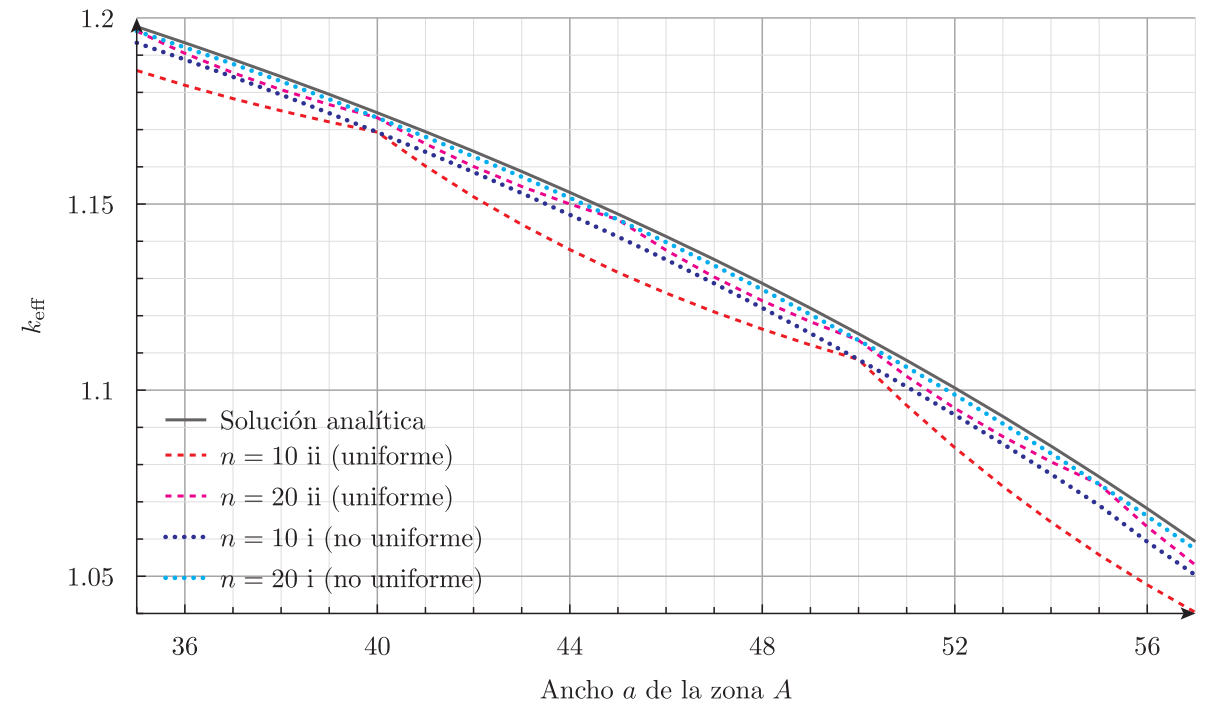
5.7. Estudios paramétricos: el reactor cubo-esferoidal

> **TL;DR**: Un "experimento pensado" para verificar que una esfera es más eficiente que un cubo. No es posible resolver una geometría con bordes curvos con una malla cartesiana estructurada, mientras que elementos curvos de segundo orden pueden discretizar superficies cónicas exactamente.

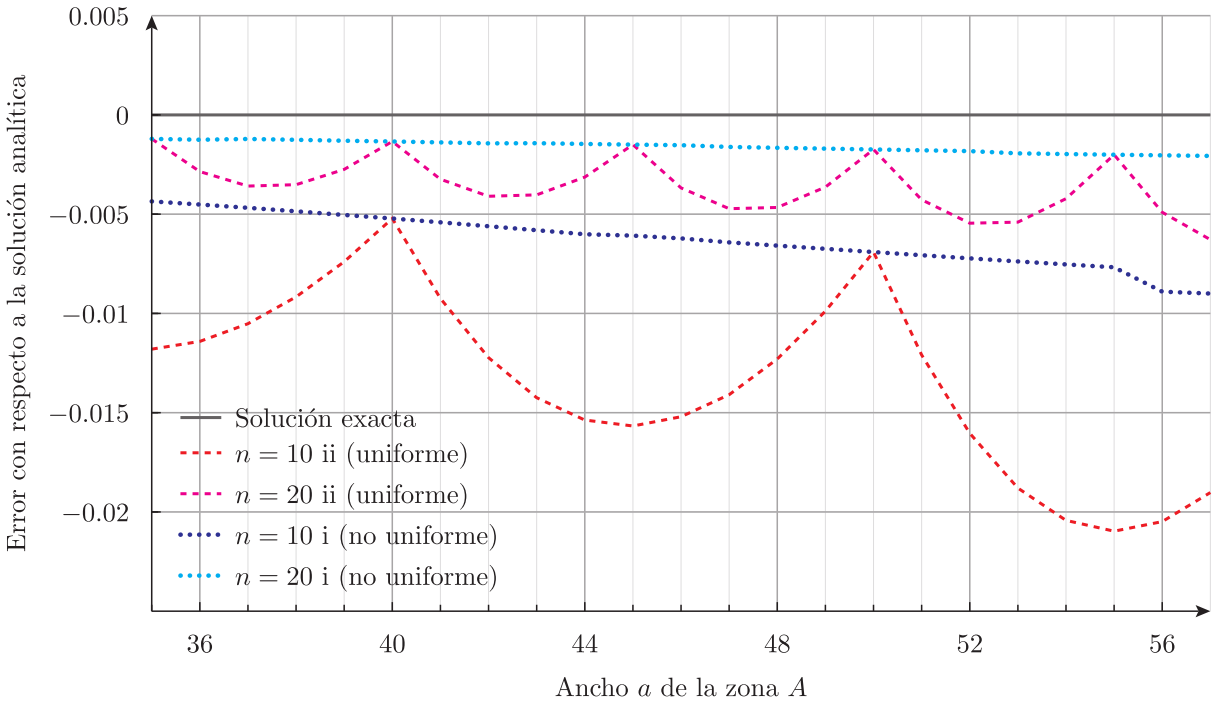
Todos sabemos que, a volumen constante, un reactor desnudo esférico tiene un factor de multiplicación efectivo mayor que un reactor cúbico (figura 5.39). De hecho a un grupo de energía con aproximación de difusión podemos calcular explícitamente dicho k_{eff}.

Pero, ¿qué pasa a medida que el cubo se va transformando en una esfera? Es decir, ¿cómo cambia k_{eff} a medida que redondeamos los bordes del cubo como en la figura 5.40? Trabajo para Gmsh & FeenoX. Comencemos con el archivo de entrada de FeenoX, que es realmente sencillo:

```
PROBLEM neutron_diffusion DIMENSIONS 3
READ_MESH cubesphere-$1.msh DIMENSIONS 3

D1 = 1.03453E+00
Sigma_a1 = 5.59352E-03
nuSigma_f1 = 6.68462E-03
Sigma_s1.1 = 3.94389E-01

PHYSICAL_GROUP fuel DIM 3
BC internal    mirror
BC external    vacuum
```

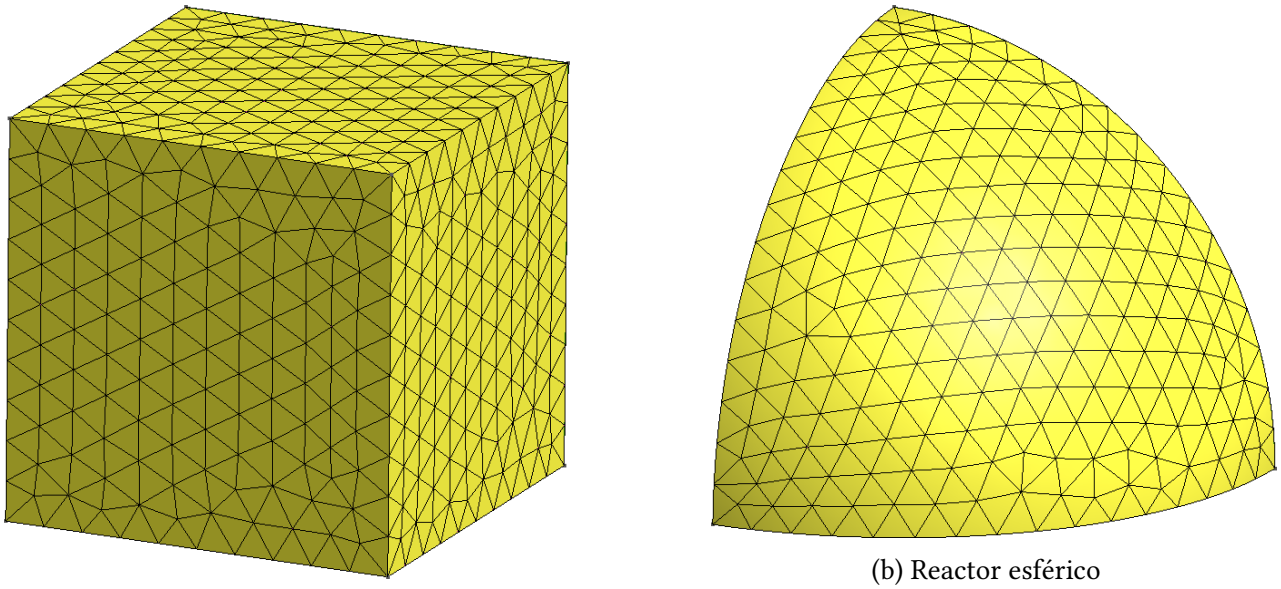


(a) Factor de multiplicación



(b) Error con respecto a la solución analítica

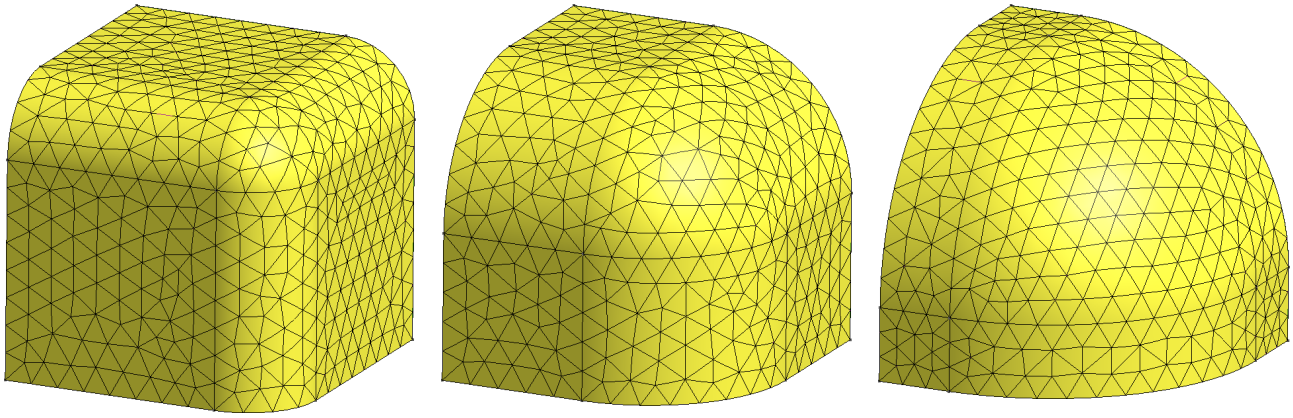
Figura 5.38.: Efecto "cúspide": comparación entre los k_{eff} obtenidos en los puntos *i* y *ii* con respecto a la solución analítica.



(a) Reactor cúbico

(b) Reactor esférico

Figura 5.39.: Un octavo de dos reactores desnudos



(a) 75% cubo/25% esfera

(b) 50% cubo/50% esfera

(c) 25% cubo/75% esfera

Figura 5.40.: Transformación continua entre un cubo y una esfera

```
SOLVE_PROBLEM

PRINT HEADER $1 keff 1e5*(keff-1)/keff fuel_volume
```

En efecto, la mayor complejidad está en generar la malla para una fracción $0 < f < 1$ de "esfericidad" (de forma que $f = 0$ sea un cubo y $f = 1$ sea una esfera) para un volumen V constante. Dado que hay cuestiones no triviales, como por ejemplo el hecho de que el kernel geométrico OpenCASCADE no permite hacer una operación tipo "fillet" de forma tal que se eliminen las caras planas originales (que es justamente lo que sucede para $f = 1$) es mejor generar la malla usando la interfaz Python de Gmsh en lugar de un archivo de entrada:

```
import os
import math
import gmsh

def create_mesh(vol, f100):
  gmsh.initialize()
  gmsh.option.setNumber("General.Terminal", 0)

  f = 0.01*f100
  # cuánto tiene que valer a para mantener el volumen constante?
  a = (vol / (1/8*4/3*math.pi*f**3 + 3*1/4*math.pi*f**2*(1-f) + 3*f*(1-f)**2 + (1-f)**3))**(1.0/3.0)

  internal = []
  gmsh.model.add("cubesphere")
  if (f100 < 1):
    # a cube
    gmsh.model.occ.addBox(0, 0, 0, a, a, a, 1)
    internal = [1,3,5]
    external = [2,4,6]

  elif (f100 > 99):
    # a sphere
    gmsh.model.occ.addSphere(0, 0, 0, a, 1, 0, math.pi/2, math.pi/2)
    internal = [2,3,4]
    external = [1]

  else:
    gmsh.model.occ.addBox(0, 0, 0, a, a, a, 1)
    gmsh.model.occ.fillet([1], [12, 7, 6], [f*a], True)
    internal = [1,4,6]
    external = [2,3,5,7,8,9,10]

  gmsh.model.occ.synchronize()

  gmsh.model.addPhysicalGroup(3, [1], 1)
  gmsh.model.setPhysicalName(3, 1, "fuel")
  gmsh.model.addPhysicalGroup(2, internal, 2)
  gmsh.model.setPhysicalName(2, 2, "internal")
  gmsh.model.addPhysicalGroup(2, external, 3)
  gmsh.model.setPhysicalName(2, 3, "external")

  gmsh.option.setNumber("Mesh.ElementOrder", 2)
  gmsh.option.setNumber("Mesh.Optimize", 1)
  gmsh.option.setNumber("Mesh.OptimizeNetgen", 1)
  gmsh.option.setNumber("Mesh.HighOrderOptimize", 1)
  gmsh.option.setNumber("Mesh.CharacteristicLengthMin", a/10);
  gmsh.option.setNumber("Mesh.CharacteristicLengthMax", a/10);
```

```
  gmsh.model.mesh.generate(3)
  gmsh.write("cubesphere-%g.msh"%(f))
  gmsh.finalize()
  return


for f100 in range(0,101,5):
  create_mesh(100**3, f100)
  os.system("feenox cubesphere.fee %g"%(f100))
```

Al ejecutar directamente este script de Python tenemos el k_{eff} en función de f (la última columna muestra el volumen de la malla para verificar que estamos manteniéndolo razonablemente constante para todos los valores de f) que graficamos en la figura 5.41:

```
$ python cubesphere.py | tee cubesphere.dat
0       1.05626 5326.13 1e+06
5       1.05638 5337.54 999980
10      1.05675 5370.58 999980
15      1.05734 5423.19 999992
20      1.05812 5492.93 999995
25      1.05906 5576.95 999995
30      1.06013 5672.15 999996
35      1.06129 5775.31 999997
40      1.06251 5883.41 999998
45      1.06376 5993.39 999998
50      1.06499 6102.55 999998
55      1.06619 6208.37 999998
60      1.06733 6308.65 999998
65      1.06839 6401.41 999999
70      1.06935 6485.03 999998
75      1.07018 6557.96 999998
80      1.07088 6618.95 999998
85      1.07143 6666.98 999999
90      1.07183 6701.24 999999
95      1.07206 6721.33 999998
100     1.07213 6727.64 999999
$
```

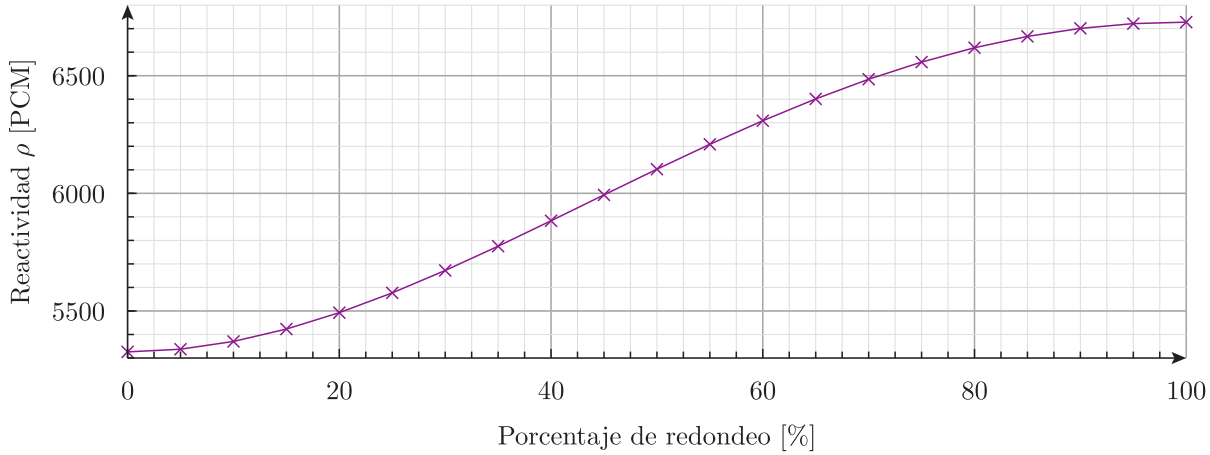


Figura 5.41.: Reactividad estática vs. fracción de esfericidad f

5.8. Optimización: el problema de los pescaditos

> **TL;DR**: Estudios paramétricos y de optimización al estilo Unix.

Los tres problemas de esta sección—inventados para escribir el artículo [64]—ilustran cómo generalizar las facilidades que provee FeenoX para realizar estudios paramétricos (tal como ya hemos hecho en casi todos los problemas anteriores) para dar un paso más allá y resolver un problema de optimización. Para eso comenzamos resolviendo numéricamente "el problema del pescadito[9]", que es un clásico en la teoría de perturbaciones lineales. Agregamos luego un segundo pescadito que nada en forma diametralmente opuesta al primero donde mostramos cómo aparecen los dos efectos de apantallamiento y anti-apantallamiento según las posiciones relativas de ambos pescaditos. Pasamos finalmente a dejar dos pescaditos fijos y resolver la pregunta: ¿dónde tenemos que poner un tercer pescadito para que la reactividad neta sea mínima?

Observación. El problema "tradicional" de los pescaditos en tri-dimensional. En esta sección nos quedamos con la versión en dos dimensiones para simplificar el tratamiento—especialmente en la sección 5.8.3—pero el espíritu es el mismo.

Observación. En los dos primeros problemas podríamos explotar la simetría para reducir el tamaño del problema pero preferimos resolver la geometría completa para ilustrar mejor el problema. Ya hemos discutido la reducción del número total de grados de libertad en la sección 5.3.

5.8.1. Un pescadito: teoría de perturbaciones lineales

Consideremos un reactor bi-dimensional circular de radio A con centro en el origen del plano x-y y con secciones eficaces macroscópicas homogéneas a un grupo de energías. Supongamos que un pescadito circular absorbente de radio a puede moverse a lo largo del eje x positivo. Un ejercicio clásico de análisis de reactores es dar la reactividad negativa introducida por el pescadito en función de la posición $x = r$ de su centro utilizando teoría de perturbaciones (es decir, suponiendo que $a \ll A$.

En esta sección vamos a obtener con FeenoX la reactividad neta introducida por el pescadito variando paramétricamente la posición $r \in [0 : A - a]$ de su centro $x = r$, $y = 0$. Para eso, comenzamos con un script de Bash que…

1. genera un archivo de texto `vars.geo` (que es incluido desde `un-pescadito.geo` para ubicar el círculo pequeño correspondiente al pescadito),
2. crea una malla no estructurada con Gmsh (figura 5.43)
3. resuelve la ecuación de difusión con FeenoX para diferentes posiciones r del pescadito.

```
#!/bin/bash

A=50   # radio del reactor
a=2    # radio del pescadito
```

[9] Decidimos usar la palabra *pescadito* en lugar de *pececito* ya que es mucho más simpática. Además, es poco probable que un pez pueda sobrevivir nadando en un reactor líquido homogéneo de sales fundidas.

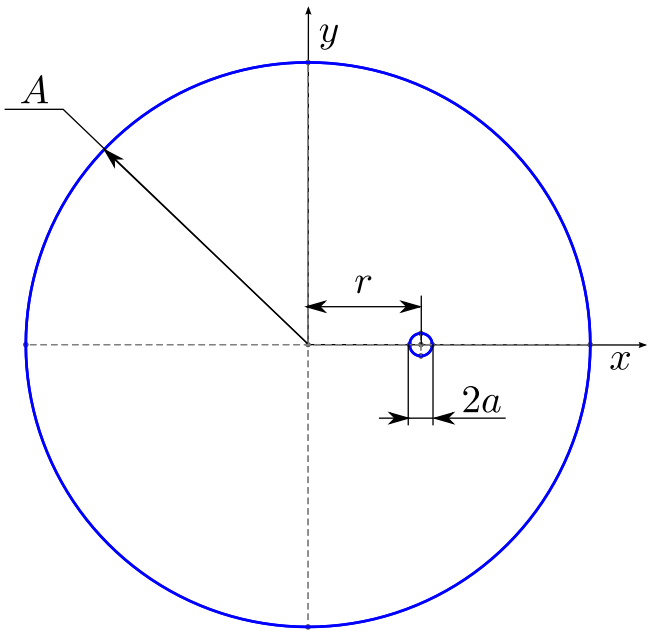
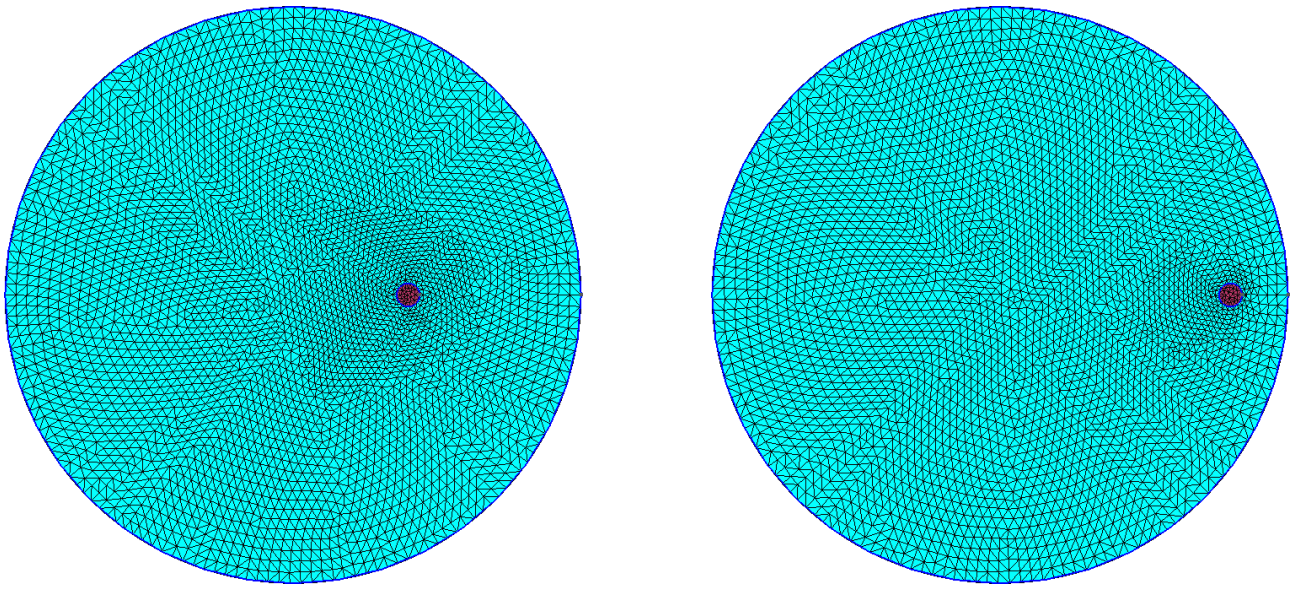


Figura 5.42.: Un pescadito de radio a nada a lo largo del eje x en un reactor homogéneo de radio A.



(a) $r = 20$ (b) $r = 40$

Figura 5.43.: Mallas para un pescadito rojo de radio $a = 2$ nadando dos posiciones $x = r$, $y = 0$ en un reactor celeste de radio $A = 50$.

```
rm -f tmp
for r in $(feenox steps.fee 0 47 64); do
 cat << EOF > vars.geo
A = ${A};
a = ${a};
r = ${r};
EOF
 gmsh -2 -v 0 un-pescadito.geo
 feenox un-pescadito.fee | tee -a tmp
done
sort -g tmp > un-pescadito.csv
```

Observación. Notar que a diferencia de los estudios paramétricos realizados hasta el momento que involucraban una serie creciente de valores, en este caso el bucle `for` de Bash se realiza sobre la salida del archivo de FeenoX `steps.fee`:

```
# steps.fee: genera secuencias de números
DEFAULT_ARGUMENT_VALUE 1 0   # min
DEFAULT_ARGUMENT_VALUE 2 1   # max
DEFAULT_ARGUMENT_VALUE 3 1   # steps
DEFAULT_ARGUMENT_VALUE 4 0   # offset
static_steps = $4+$3
x = quasi_random($1,$2)
IF (step_static>($4+0.1))
 PRINT x
ENDIF
```

que genera una sucesión (determinística) de números cuasi-aleatorios que, eventualmente, llena densamente un hipercubo [55]. En los lazos paramétricos crecientes, si se desea aumentar la densidad del barrido hay que volver a calcular todo el intervalo nuevamente. En una serie de números cuasi-aleatorios, es posible agregar nuevos puntos a los ya calculados dando un *offset* inicial. En la sección 5.11 esta característica de las serie cuasi-aleatorias es más evidente ya que allí barremos densamente un espacio de parámetro bi-dimensional. Por ejemplo, podemos barrer el intervalo $[0, 1]$ con tres puntos con el archivo `steps.fee` como

```
$ feenox steps.fee 0 1 3
0.5
0.75
0.25
$
```

Si luego del estudio quisiéramos agregar cuatro puntos más, comenzamos con el offset 3 para obtener

```
$ feenox steps.fee 0 1 4 3
0.375
0.875
0.625
0.125
$
```

El archivo de entrada de Gmsh genera dos círculos y asigna etiquetas para materiales y condiciones de contorno:

```
Merge "vars.geo";

SetFactory("OpenCASCADE");
Disk(1) = {0, 0, 0, A};
Disk(2) = {r, 0, 0, a};
Coherence;

Physical Curve("external") = {1};
Physical Surface("fuel") = {3};
Physical Surface("pescadito") = {2};

Color Cyan      { Surface{3}; }
Color Maroon    { Surface{2}; }

Mesh.Algorithm = 9;
Mesh.MeshSizeMax = A/25;
Mesh.MeshSizeFromCurvature = 16;
Mesh.Optimize = 1;
```

El archivo de entrada de FeenoX es muy sencillo:

```
PROBLEM neutron_diffusion GROUPS 1 DIMENSIONS 2
READ_MESH un-pescadito.msh
INCLUDE xs.fee
INCLUDE vars.geo
SOLVE_PROBLEM

# calculamos la reactividad que tendría el círculo sin pescaditos
# para eso miramos las XS en un punto donde sepamos que no va a estar el pescadito
rho0 = (nuSigma_f1(0.5*A,0.5*A) - Sigma_a1(0.5*A,0.5*A) -  ↩
    D1(0.5*A,0.5*A)*(2.4048/A)^2)/nuSigma_f1(0.5*A,0.5*A)
rho = (keff-1)/keff

# el delta rho que metió el pescado vs. r
PRINT r 1e5*(rho-rho0)
```

La única "complejidad" aparece al calcular el incremento negativo de reactividad con respecto al k_{eff} analítico del círculo original. Para ello ponemos una condición de contorno de flujo nulo en la circunferencia dentro del archivo incluido `xs.fee`:

```
# secciones eficaces
MATERIAL fuel       D1=1   Sigma_a1=0.02  nuSigma_f1=0.03
MATERIAL pescadito  D1=0.9 Sigma_a1=0.10
BC external null
```

Observación. La constante 2.4048 que aparece en el archivo de entrada de FeenoX es una aproximación del primer cero de la función $J_0(\xi)$ de Bessel.

Al ejecutar el script obtenemos la curva que mostramos en la figura 5.44: mientras más alejado del centro nade el pescadito, menor es la reactividad neta introducida.

```
$ ./un-pescadito.sh
23.5    -606.765
35.25   -202.255
11.75   -1094.41
[...]
```

```
24.2344 -576.9
0.734375        -1339.52
1.10156 -1338.06
$
```

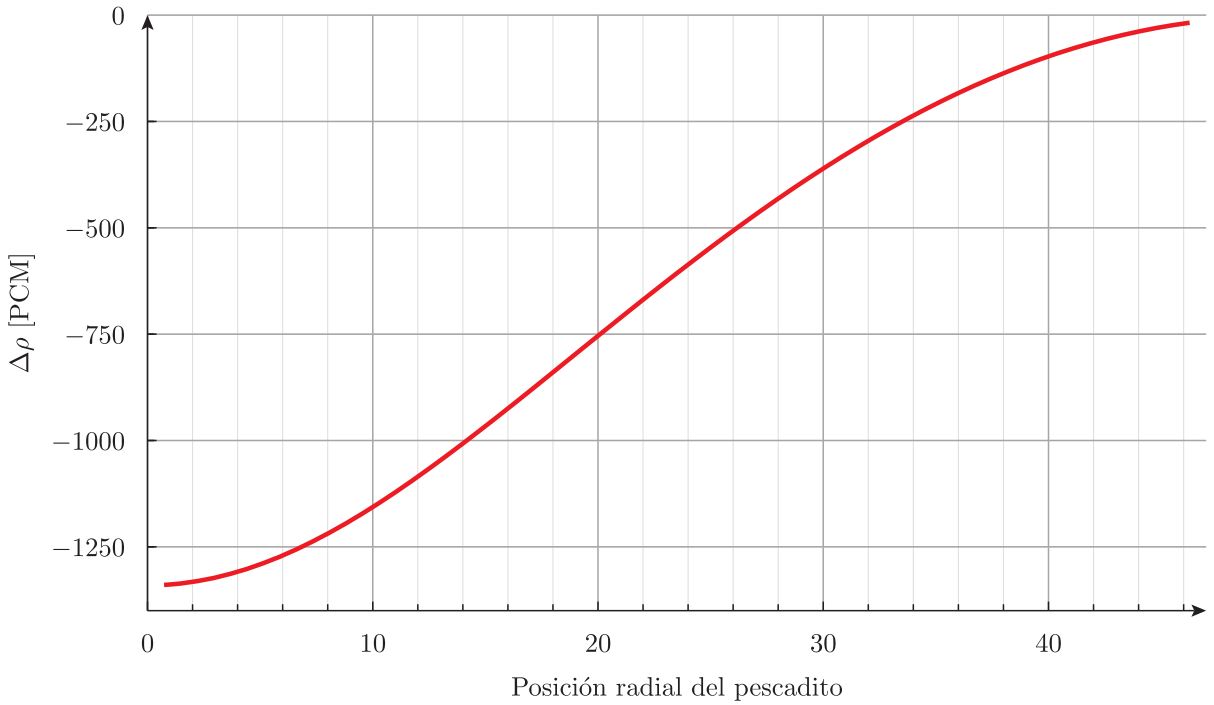


Figura 5.44.: Curva "S" de reactividad negativa introducida por un pescadito en un reactor circular.

5.8.2. Dos pescaditos: estudio paramétrico no lineal

Agreguemos ahora otro pescadito nadando en forma diametralmente opuesta al primero (figura 5.45). El script de Bash, el archivo de entrada de Gmsh y el de FeenoX son muy similares a los de la sección anterior por lo que no los mostramos. Ahora al graficar la curva de reactividad en función de r y compararla con el doble de la reactividad introducida por un único pescadito calculada en la sección anterior (figura 5.46a) vemos los dos efectos explicados en el capítulo 14 de la referencia clásica [31] (figura 5.46b). Si los pescaditos están lejos entre sí sus reactividades negativas son más efectivas porque se anti-apantallan. Pero al acercarse, se apantallan y la reactividad neta es menor que por separado.

5.8.3. Tres pescaditos: optimización

Supongamos ahora que agregamos un tercer pescadito. Por alguna razón, los primeros dos pescaditos están fijos en dos posiciones arbitrarias $[x_1, y_1]$ y $[x_2, y_2]$. Tenemos que poner el tercer pescadito en una posición $[x_3, y_3]$ de forma tal de hacer que la reactividad total sea lo menor posible.[10]

[10] Reemplazar "pescadito" por "lanza de inyección de boro de emergencia" para pasar de un problema puramente académico a un problema de interés en ingeniería nuclear.

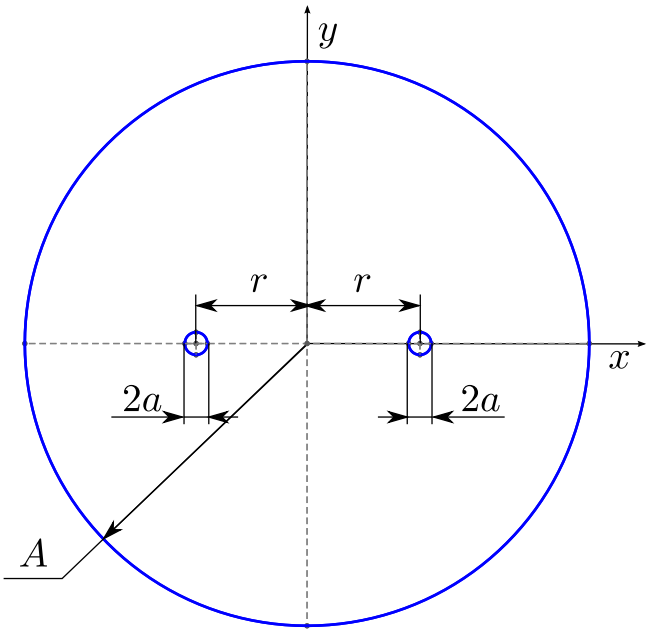


Figura 5.45.: Dos pescaditos de radio a nadan a lo largo del eje x en forma diametralmente opuesta.

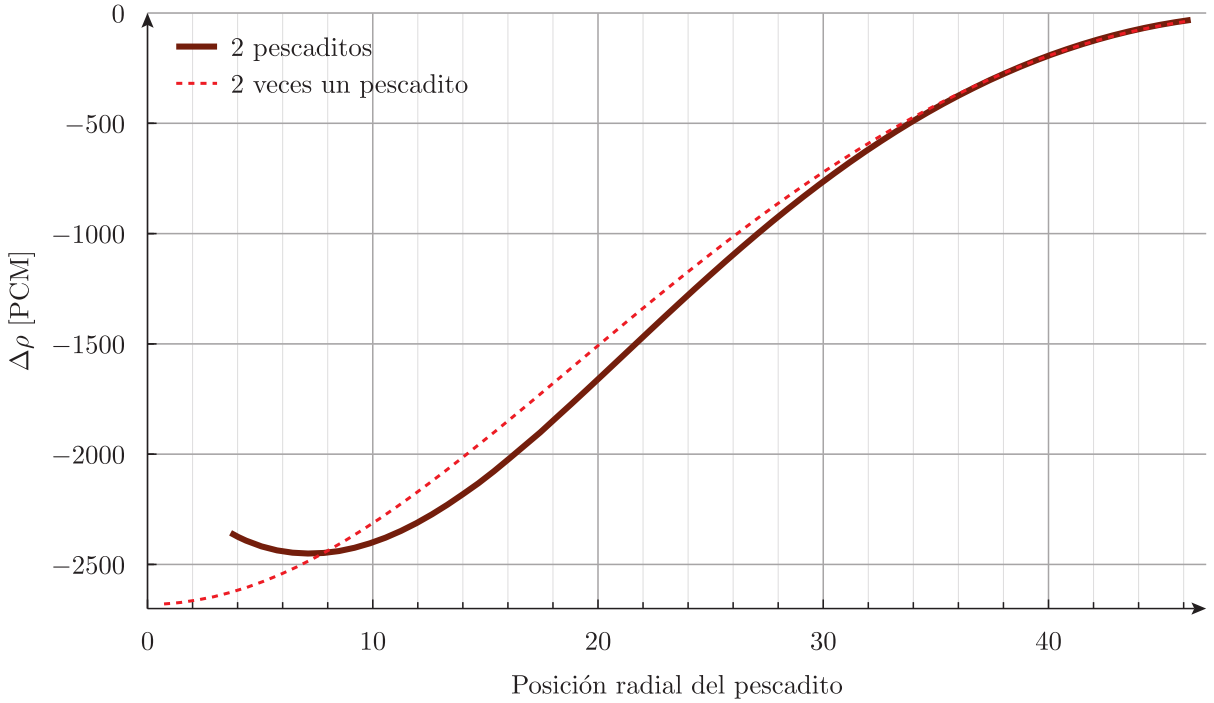
Una forma de resolver este problema con FeenoX es atacarlo en forma similar a la manera en que procedimos que en las secciones anteriores. Pero ahora en lugar de variar la posición del tercer pescadito en forma paramétrica según una receta determinística ya conocida de antemano, utilizar un algoritmo de optimización que decida la nueva posición del tercer pescadito en función de la historia de posiciones y los valores de reactividad calculados por FeenoX en cada paso.

Observación. En la reciente tesis de maestría [89] se emplea FeenoX para resolver la ecuación de calor y, mediante un script en Python, optimizar topológicamente el reflector de un reactor nuclear integrado. Esa tesis, junto con esta sección, ayuda a ilustrar el punto que queremos enfatizar sobre la flexibilidad en el diseño de FeenoX para ser utilizada como una herramienta de optimización.

En particular, podemos usar la biblioteca de Python SciPy que provee acceso a algoritmos de optimización y permite con muy pocas líneas de Python implementar el bucle de optimización:

```
#!/usr/bin/python
from scipy.optimize import minimize
import numpy as np
import subprocess

def keff(x3):
  r = subprocess.run(["bash", "tres.sh", "(%s)" % str(x3[0]), "(%s)" % str(x3[1])],  ↩
      stdout=subprocess.PIPE)
  k = float(r.stdout.decode('utf-8'))
  return k

x3_0 = [+25, -25]
result = minimize(keff, x3_0, method="nelder-mead")
print(result)
```

Observación. En forma deliberada hemos mostrado en este caso un driver script muy sencillo para mostrar que realmente es posible realizar un cálculo de optimización con muy poco código extra.



(a) Reactividad debida a dos pescaditos y el doble de la reactividad de uno solo.

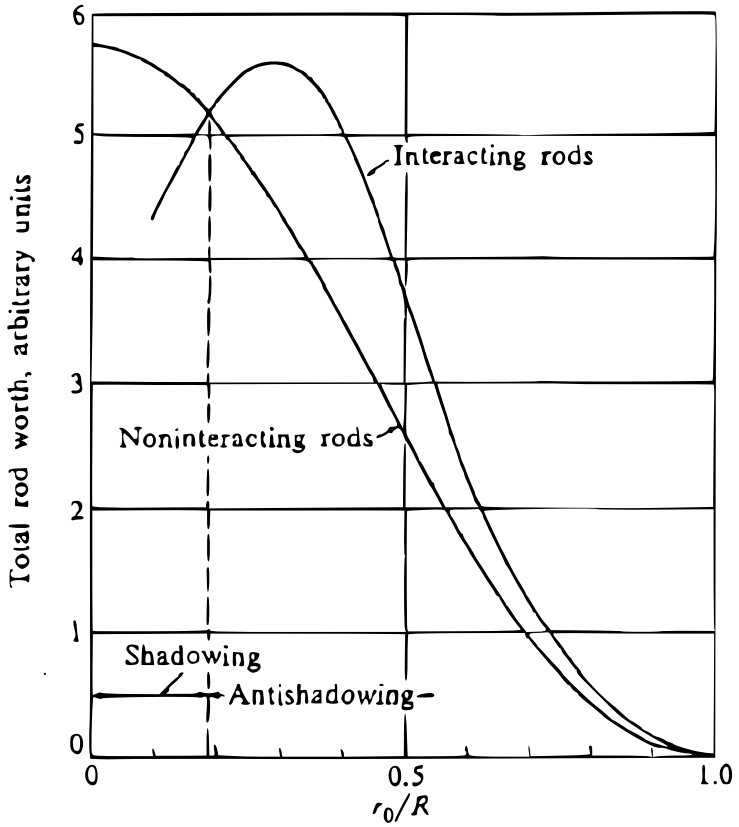


Fig. 14–5. The worth of two diametrically opposite control rods as a function of their distance from the axis of a cylindrical reactor, with and without interaction.

(b) Figura 14-5 de la referencia [31]

Figura 5.46.: Efecto de apantallamiento y anti-apantallamiento

Sin embargo, el esquema propuesto también funciona para otros algoritmos de optimización más complejos como recocido simulado, algoritmos genéticos o incluso aquellos basados en redes neuronales.

La función `keff()` a optimizar es función de la posición $\mathbf{x}_3$ del tercer pescadito, cuyas dos componentes son pasadas como argumento al script de Bash que llama primero a Gmsh y luego a FeenoX para devolver el $k_{\text{eff}}(\mathbf{x}_3)$ para $\mathbf{x}_1$ y $\mathbf{x}_2$ fijos:

```
cat << EOF > x3.geo
x3 = $1;
y3 = $2;
EOF
gmsh -2 -v 0 tres-pescaditos.geo
feenox tres-pescaditos.fee
```

```
Merge "vars.geo";

x1 = -10;    // centro del pescado 1
y1 = 5;

x2 = 25;     // centro del pescado 2
y2 = -15;

Merge "x3.geo";

SetFactory("OpenCASCADE");
Disk(1) = {0, 0, 0, A};
Disk(2) = {x1, y1, 0, a};
Disk(3) = {x2, y2, 0, a};
Disk(4) = {x3, y3, 0, a};
Coherence;

Physical Curve("external") = {1};
Physical Surface("fuel") = {5};
Physical Surface("pescadito") = {2,3,4};

Color Cyan      { Surface{5}; }
Color Maroon    { Surface{2,3,4}; }

Mesh.Algorithm = 9;
Mesh.MeshSizeMax = A/25;
Mesh.MeshSizeFromCurvature = 16;
Mesh.Optimize = 1;
```

```
PROBLEM neutron_diffusion 2D
READ_MESH tres-pescaditos.msh
INCLUDE xs.fee
SOLVE_PROBLEM

PRINT %.8f keff
```

Observación. Si FeenoX, tal como Gmsh, tuviese una interfaz Python (tarea que está planificada e incluso tenida en cuenta en la base de diseño de FeenoX) entonces la función `keff(x3)` a minimizar sería más elegante que la propuesta, que involucra hacer un `fork()+exec()` para invocar a un script de Bash que hace otros dos `fork()+exec()` para ejecutar `gmsh` y `feenox`.

La ejecución del script de optimización muestra la reactividad mínima en `fun` y la posición óptima del tercer pescadito en `x`. Podemos ver los pasos intermedios en la figura 5.47 y 5.48.

```
$ ./tres.py
 final_simplex: (array([[ 3.55380249, -1.15953827],
       [ 3.41708565, -1.71419764],
       [ 3.89513397, -1.83193016]]), array([1.29409226, 1.29409476, 1.294099  ]))
           fun: 1.29409226
       message: 'Optimization terminated successfully.'
          nfev: 37
           nit: 20
        status: 0
       success: True
             x: array([ 3.55380249, -1.15953827])
$
```

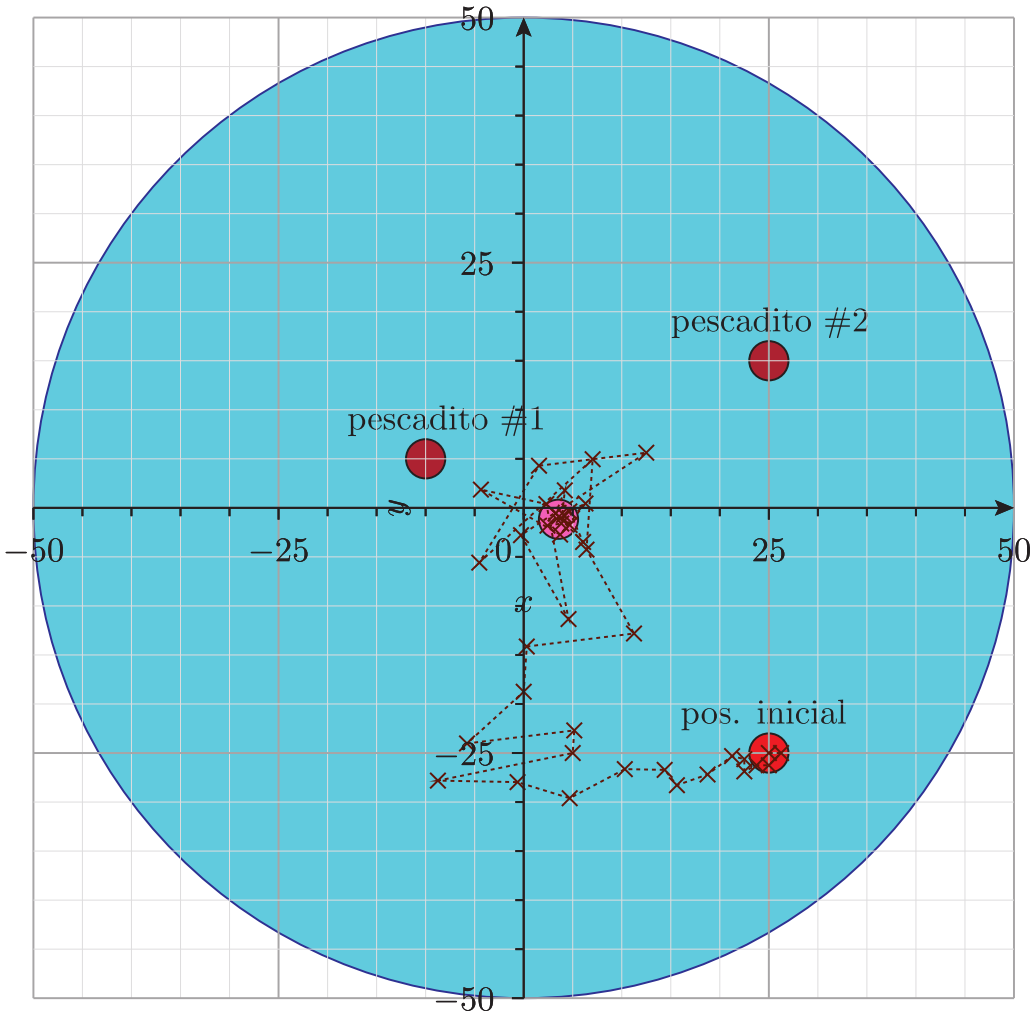


Figura 5.47.: Pasos intermedios del problema de optimización de los tres pescaditos resuelto con el algoritmo de Nelder-Mead [39].

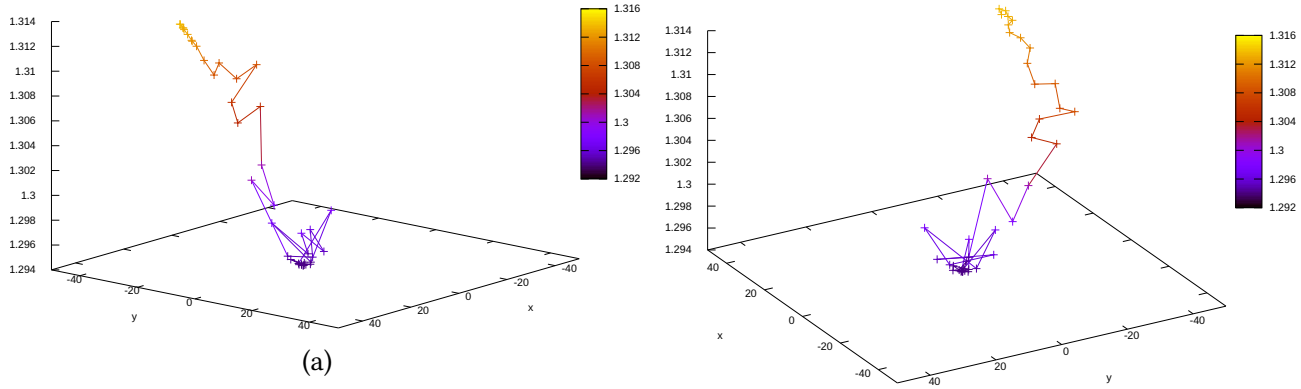


Figura 5.48.: Factor de multiplicación k_{eff} en función de la posición $\mathbf{x}_3$ para los pasos intermedios.

5.9. Verificación con el método de soluciones fabricadas

> **TL;DR**: Para verificar los algoritmos numéricos de un código con el método de soluciones fabricadas se necesita que dicho código permita definir propiedades materiales en función del espacio a través de expresiones algebraicas arbitrarias.

Como mencionamos brevemente en la sección 5.5, la verificación de códigos de cálculo involucra mostrar que dicho código resuelve correctamente las ecuaciones que debe resolver. La forma de hacerlo es calcular alguna medida del error cometido por el método numérico implementado en el código, por ejemplo el error L_2

$$e_2 = \frac{\sqrt{\int \left[\phi_{\text{num}}(\mathbf{x}) - \phi_{\text{ref}}(\mathbf{x})\right]^2 d^D\mathbf{x}]}}{\int d^D\mathbf{x}} \tag{5.2}$$

y mostrar que dicho error tiende a cero cuando el tamaño del problema discretizado tiene a infinito. Según la referencia [51],

> "The code author defines precisely what continuum partial differential equations and continuum boundary conditions are being solved, and convincingly demonstrates that they are solved correctly, i.e., usually with some order of accuracy, and always consistently, so that as some measure of discretization (e.g. the mesh increments), the code produces a solution to the continuum equations; this is Verification."

Más aún, según la referencia [53] dice

> ... we recommend that, when possible, one should demonstrate that the equations are solved to the theoretical order-of-accuracy of the discretization method.

Esto quiere decir que la forma de tender a cero debe coincidir con el orden predicho por la teoría. Para difusión de neutrones con elementos finitos, este orden es 2 para elementos de primer orden y 3 para elementos de segundo orden. La forma de hacer esto es

1. Resolver el problema para un cierto tamaño característico h de malla.
2. Calcular el error e_2 (o alguna otra medida del error) en función de h.
3. Verificar que la pendiente de e_2 vs. h en un gráfico log-log es la predicha por la teoría.

De todas maneras, para mostrar que el error tiende a cero necesitamos tener una expresión algebraica para la solución exacta de la ecuación que queremos resolver. Es decir, necesitamos conocer $\phi_{\text{ref}}(\mathbf{x})$ en la ecuación 5.2. Está claro que si conociéramos esta expresión para un caso general, esta tesis no tendría razón de ser. Y es razonable que no exista esta expresión porque, de alguna manera, resolver la ecuación de difusión de neutrones involucra "integrar" dos veces la fuente de neutrones.

El método de soluciones fabricadas (o MMS por sus siglas en inglés) propone recorrer el camino inverso: partir de una solución conocida (es decir, fabricada *ad hoc*) y preguntarnos cuál es la fuente necesaria para dar lugar a ese flujo. Este camino es mucho más sencillo ya que involucra "derivar" la fuente dos veces, y es el método que ilustramos en esta sección.

Por otro lado, [53] también dice que hay que asegurarse "probar" todas las características del software, incluyendo

- condiciones de contorno
- tipos de elementos
- solvers algebraicos
- modelos de materiales
- etc.

lo que rápidamente da lugar a una explosión combinatoria de parámetros como la que apareció en la sección 5.4. Siguen diciendo los autores de [53]

> To ensure that all code options relevant to code Verification are tested, one must design a suite of coverage tests. Fortunately, this is not as daunting as it may seem at first. If a code has options, the number of coverage tests needed to verify the code is determined by the number of mutually exclusive options, i.e., there is no combinatorial explosion of tests to run. For example, suppose a code has two solver options and three constitutive relationship (CR) options. Then only three coverage tests are needed to check all options. Test (1): Solver 1 with CR1, Test (2): Solver 2 with CR2,Test (3): Solver 1 or 2 with CR3. One does not need to perform a test involving the combination, for example, Solver 1 and CR2 because Test 1 will ascertain whether of not Solver 1 is working correctly, while Test 2 will ascertain whether or not CR2 is working correctly.

En mi experiencia personal, la explosión combinatoria *sí* existe. La aplicación de este método al problema de conducción de calor es objeto de estudio de un informe técnico [67] y una presentación a un congreso [69]. En las dos secciones que siguen lo aplicamos al problema de difusión de neutrones, primero a un grupo con secciones eficaces uniformes en tres dimensiones para introducir la idea. Luego en la sección 5.9.2 a dos grupos con secciones eficaces dependientes del espacio en dos dimensiones.

Observación. Para verificar un código con el método de MMS es condición necesaria que el código permita definir fuentes a través de expresiones algebraicas. Es también recomendable que permita realizar estudios paramétricos con cierta facilidad y automatización razonable [70].

5.9.1. Stanford bunny a un grupo

Consideremos un reactor con la forma del conejo de Stanford (figura 5.49) con secciones eficaces adimensionales uniformes $D = 1$ y $\Sigma_a = 0.05$. La ecuación de difusión es entonces

Figura 5.49.: El famoso y nunca bien ponderado conejo de Stanford

$$-\text{div}\left[D \cdot \text{grad}(\phi)\right] + \Sigma_a \cdot \phi(x,y,z) = S(x,y,z) \tag{5.3}$$

Propongamos una solución fabricada para el flujo escalar, digamos

$$\phi(x,y,z) = \log\left[1 + 10^{-2} \cdot z \cdot (y+50)\right] + 10 \cdot (x + \sqrt{y+50}) \tag{5.4}$$

La fuente que necesitamos proviene de reemplazar la ecuación 5.4 en la ecuación 5.3. Siguiendo las reglas de generación y de composición de Unix lo que hacemos es usar Maxima, que es un sistema para la manipulación simbólica y numérica de expresiones libre, abierto y ameno a la filosofía Unix, para que haga las cuentas por nosotros. En efecto, consideremos el siguiente archivo de entrada:

```
phi1_mms(x,y,z) = log(1+1e-2*z*(y+50)) + 1e-3*(x+sqrt(y+50))
D1(x,y,z) = 1
Sigma_a1(x,y,z) = 0.05

DEFAULT_ARGUMENT_VALUE 1 tet4       # elemento = tet4/tet10
DEFAULT_ARGUMENT_VALUE 2 16         # factor de refinamiento
DEFAULT_ARGUMENT_VALUE 3 0          # escribimos vtk? = 0/1
```

```
READ_MESH bunny-$1-$2.msh
PROBLEM neutron_diffusion 3D

# incluimos la fuente calculada con maxima
INCLUDE neutron-bunny-s.fee

# ponemos condicion de contorno de dirichlet
BC external phi1=phi1_mms(x,y,z)

SOLVE_PROBLEM

# salidas
PHYSICAL_GROUP bunny DIM 3
h = (bunny_volume/cells)^(1/3)

# L-2 error
INTEGRATE (phi1(x,y,z)-phi1_mms(x,y,z))^2 RESULT e_2
error_2 = sqrt(e_2)/bunny_volume

# L-inf error
FIND_EXTREMA abs(phi1(x,y,z)-phi1_mms(x,y,z)) MAX error_inf

PRINT %.6f log(h) log(error_inf) log(error_2) %g $2 cells nodes %.2f 1024*memory() wall_time()

IF $3
  WRITE_MESH neutron-bunny_$1-$2-$3.vtk phi1 phi1_mms phi1(x,y,z)-phi1_mms(x,y,z)
ENDIF
```

De hecho, dado que la sintaxis del parser algebraico de Maxima es (casi) la misma que la de FeenoX, podemos tomar la cadena con la solución fabricada del archivo de entrada de FeenoX, calcular la fuente con Maxima y generar un archivo válido de FeenoX que podemos incluir desde el archivo de entrada principal:

```
# primero leemos el flujo y las XSs del input de FeenoX
phi1_mms=$(grep "phi1_mms(x,y,z) =" neutron-bunny.fee | sed 's/=/:=/')
D1=$(grep "D1(x,y,z) =" neutron-bunny.fee | sed 's/=/:=/')
Sigma_a1=$(grep "Sigma_a1(x,y,z) =" neutron-bunny.fee | sed 's/=/:=/')

# y después le pedimos a maxima que haga las cuentas por nosotros
maxima --very-quiet << EOF > /dev/null
${phi1_mms};
${D1};
${Sigma_a1};
s1(x,y,z) := -(diff(D1(x,y,z) * diff(phi1_mms(x,y,z), x), x) + diff(D1(x,y,z) *  ↩
    diff(phi1_mms(x,y,z), y), y) + diff(D1(x,y,z) * diff(phi1_mms(x,y,z), z), z)) +  ↩
    Sigma_a1(x,y,z)*phi1_mms(x,y,z);
stringout("neutron-bunny-s1.txt", s1(x,y,z));
tex(s1(x,y,z), "neutron-bunny-s1.tex");
EOF
```

No sólo le pedimos a Maxima que nos genere el archivo de entrada de FeenoX sino que también le podemos pedir que nos dé los macros TeX [28] para documentar la fuente en esta tesis:

$$
\begin{aligned}
S(x,y,z) =& 0.05 \left(\log\left(0.01 \ (y+50) \ z + 1\right) + 0.001 \left(\sqrt{y+50} + x \right) \right) \\
&+ \frac{1.0 \times 10^{-4} \ z^2}{\left(0.01 \ (y+50) \ z + 1\right)^2} + \frac{1.0 \times 10^{-4} \ (y+50)^2}{\left(0.01 \ (y+50) \ z + 1\right)^2} + \frac{2.5 \times 10^{-4}}{(y+50)^{\frac{3}{2}}}
\end{aligned}
$$

Lo siguiente es completar el script de Bash para generar las mallas apropiadas y graficar los resultados automáticamente:

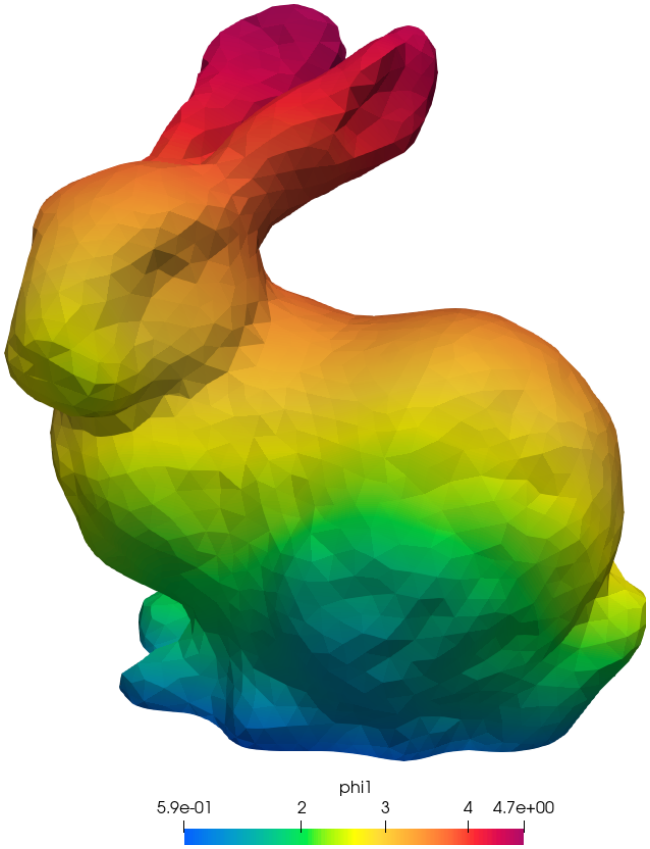
```
$ ./run.sh
# manufactured solution (input)
phi1_mms(x,y,z) := log(1+1e-2*z*(y+50)) + 1e-3*(x+sqrt(y+50));
D1(x,y,z) := 1;
Sigma_a1(x,y,z) := 0.05;

# source term (output)
S1(x,y,z) = 0.05*(log(0.01*(y+50)*z+1)+0.001*(sqrt(y+50)+x))+(1.0E-4*z^2)/(0.01*(y+50)*z+1) ↩
    ^2+(1.0E-4*(y+50)^2)/(0.01*(y+50)*z+1)^2+2.5E-4/(y+50)^(3/2)
neutron_bunny_tet4
-----------------------------------------------------------
0.882892        -2.913390       -11.628276      20      19682   4707    83.33   1.81
0.713751        -3.078327       -11.919410      24      32742   7373    92.86   2.49
0.566854        -3.515901       -12.236276      28      50921   10955   108.16  2.81
0.324435        -2.603344       -12.683721      36      105483  21355   146.90  5.16
0.224405        -3.864497       -12.850835      40      142438  28201   172.97  6.03
0.131075        -2.897239       -13.077100      44      188504  36661   204.51  7.86
0.045617        -2.848969       -13.264638      48      243633  46614   248.88  10.02
-0.068898       -3.793360       -13.505172      54      343580  64510   320.61  14.57
-0.172894       -3.198078       -13.685647      60      469441  86726   402.82  21.13
-0.236015       -3.197992       -13.823714      64      567349  103924  472.32  25.57
neutron_bunny_tet10
-----------------------------------------------------------
1.089532        -4.742025       -13.429947      16      10558   17898   100.68  1.16
0.882892        -3.691744       -14.159585      20      19682   31819   132.57  1.80
0.713751        -5.421880       -14.599096      24      32742   51285   169.52  3.20
0.566854        -5.350408       -15.058052      28      50921   77891   223.38  3.79
0.439039        -6.402456       -15.381134      32      74757   112362  304.81  4.98
0.324435        -4.280207       -15.699625      36      105483  156399  393.84  7.43
0.224405        -5.660364       -15.980139      40      142438  208860  509.98  10.78
0.131075        -3.946231       -16.107267      44      188504  273940  646.90  13.82
0.045617        -3.738939       -16.333378      48      243633  351128  807.37  18.35
$ pyxplot neutron-bunny.ppl
$
```

Observación. Aún para mallas relativamente gruesas como la de la figura 5.50, la diferencia entre el flujo numérico y la solución manufacturada es muy pequeña para ser observada a simple vista. Es necesario calcular la integral de los errores y ajustar el orden de convergencia para realmente verificar el código.

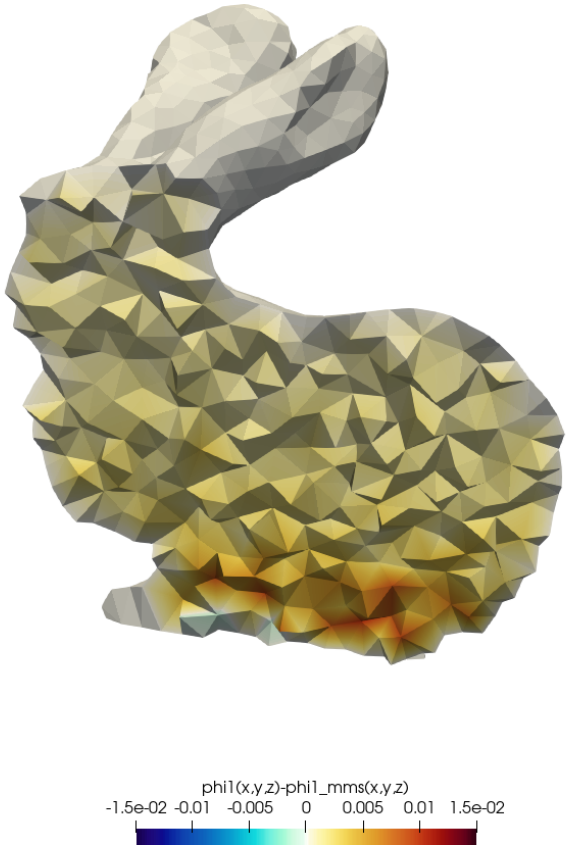
La figura 5.51 ilustra por qué el tamaño del elemento h no es una buena medida de la precisión de una discretización: obviamente, para un mismo tamaño de celda, la precisión obtenida por los elementos de mayor orden es, justamente, mucho mayor. Es por eso que es más apropiado hablar de la cantidad total de incógnitas, de grados de libertad o tamaño del problema.

5.9.2. Cuadrado a dos grupos

Simplifiquemos un poco la geometría para poder introducir complejidades en la matemática. Tomemos como geometría el cuadrado $[0:1] \times [0:1]$, pero estudiemos el problema a dos grupos con el

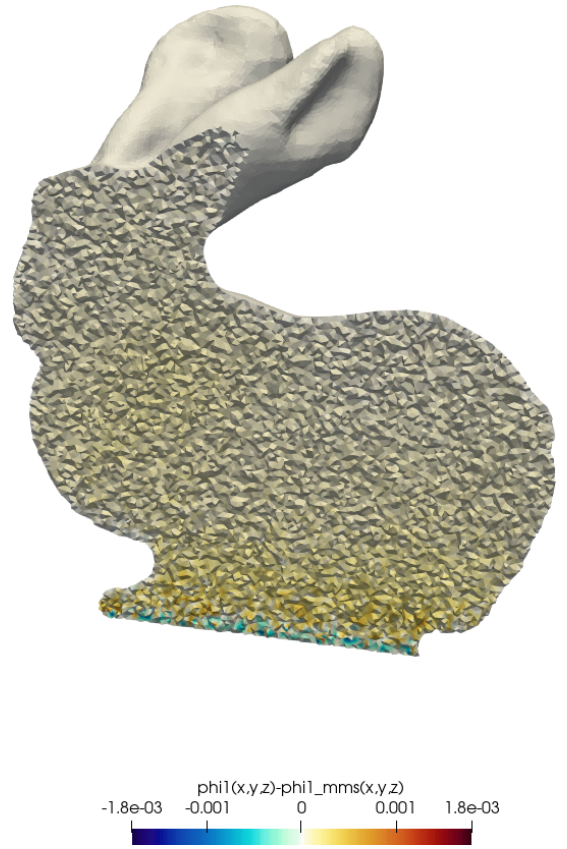


(a) $\phi(x, y, z)$ para $c = 20$



(b) $\phi_{\text{num}}(\mathbf{x}) - \phi_{\text{ref}}(\mathbf{x})$ para $c = 20$

(c) $\phi(x, y, z)$ para $c = 100$



(d) $\phi_{\text{num}}(\mathbf{x}) - \phi_{\text{ref}}(\mathbf{x})$ para $c = 100$

Figura 5.50.: Flujo y error en el conejo de Stanford para diferentes refinamientos (tet4)

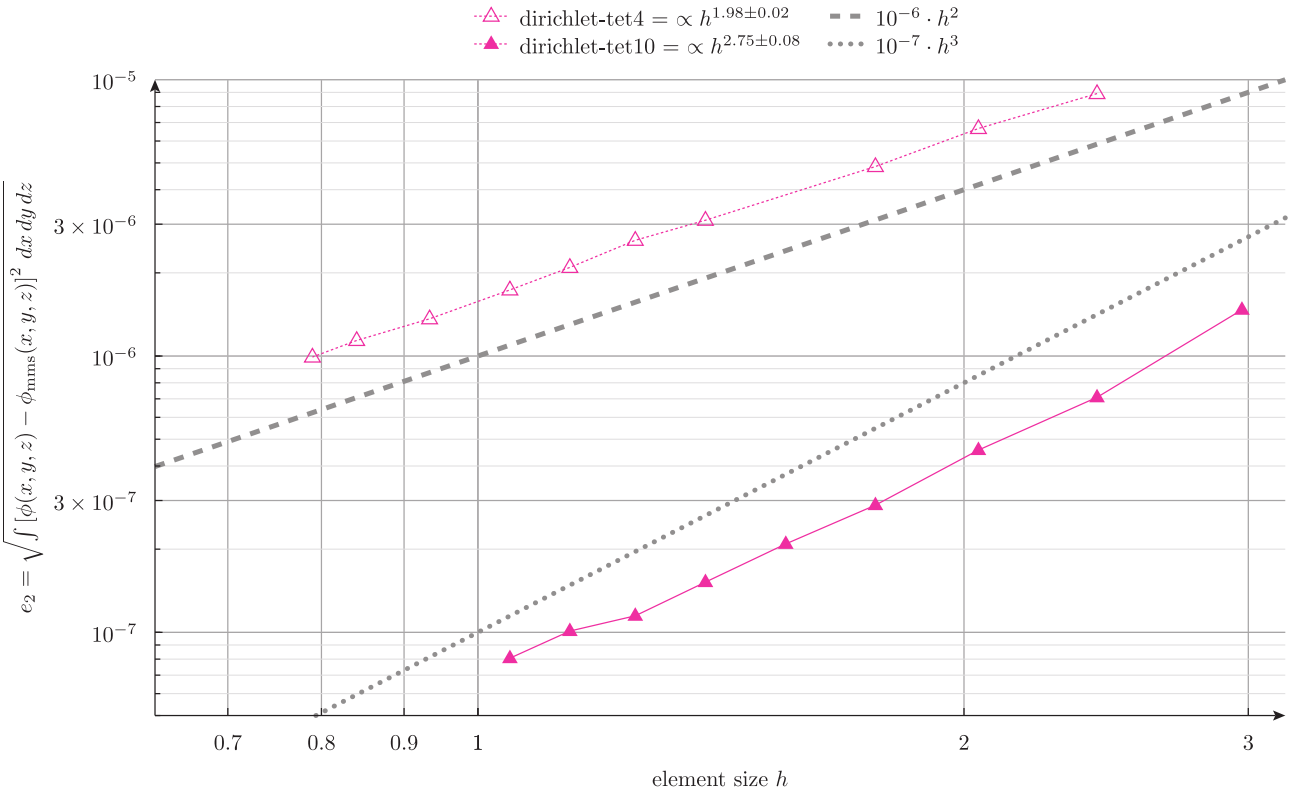


Figura 5.51.: Error e_2 vs. tamaño del elemento h para el conejo de Stanford

siguiente juego de secciones eficaces:

```
phi1_mms(x,y) = 1 + sin(2*x)^2 * cos(3*y)^2
D1(x,y) = 1 + 0.1*(x - 0.5*y)
Sigma_a1(x,y) = 1e-3*(1 + log(1+x) - 0.5*y^3)
Sigma_s1_2(x,y) = 1e-3*(1 - x + sqrt(0.5*y))

phi2_mms(x,y) = (1-0.5*tanh(-y))*log(1+x)
D2(x,y) = 1
Sigma_a2(x,y) = 1e-3
Sigma_s2_1(x,y) = 0
```

Como ahora la geometría es más sencilla, además de fijar solamente condiciones de Dirichlet en los cuatro lados del cuadrado vamos a poner dos condiciones de Dirichlet y dos de Neumann. Necesitamos entonces—además de las dos fuentes volumétricas $S1(x,y)$ y $S2(x,y)$—que Maxima nos calcule las dos componentes de las dos corrientes para que las podamos usar como condiciones de contorno:

```
phi1_mms=$(grep "phi1_mms(x,y) =" neutron-square.fee | sed 's/=/:=/')
phi2_mms=$(grep "phi2_mms(x,y) =" neutron-square.fee | sed 's/=/:=/')

D1=$(grep "D1(x,y) =" neutron-square.fee | sed 's/=/:=/')
Sigma_a1=$(grep "Sigma_a1(x,y) =" neutron-square.fee | sed 's/=/:=/')
Sigma_s1_2=$(grep "Sigma_s1_2(x,y) =" neutron-square.fee | sed 's/=/:=/')

D2=$(grep "D2(x,y) =" neutron-square.fee | sed 's/=/:=/')
Sigma_a2=$(grep "Sigma_a2(x,y) =" neutron-square.fee | sed 's/=/:=/')
Sigma_s2_1=$(grep "Sigma_s2_1(x,y) =" neutron-square.fee | sed 's/=/:=/')
```

```
maxima --very-quiet << EOF > /dev/null
${phi1_mms};
${phi2_mms};
${D1};
${Sigma_a1};
${Sigma_s1_2};
${D2};
${Sigma_a2};
${Sigma_s2_1};
s1(x,y) := -(diff(D1(x,y) * diff(phi1_mms(x,y), x), x) + diff(D1(x,y) * diff(phi1_mms(x,y), y),
    y)) + Sigma_a1(x,y)*phi1_mms(x,y) - Sigma_s2_1(x,y)*phi2_mms(x,y);
s2(x,y) := -(diff(D2(x,y) * diff(phi2_mms(x,y), x), x) + diff(D2(x,y) * diff(phi2_mms(x,y), y),
    y)) + Sigma_a2(x,y)*phi2_mms(x,y) - Sigma_s1_2(x,y)*phi1_mms(x,y);
stringout("neutron-square-s1.txt", s1(x,y));
stringout("neutron-square-s2.txt", s2(x,y));
stringout("neutron-square-j1x.txt", -D1(x,y) * diff(phi1_mms(x,y),x));
stringout("neutron-square-j1y.txt", -D1(x,y) * diff(phi1_mms(x,y),y));
stringout("neutron-square-j2x.txt", -D2(x,y) * diff(phi2_mms(x,y),x));
stringout("neutron-square-j2y.txt", -D2(x,y) * diff(phi2_mms(x,y),y));
EOF
```

El archivo de entrada de FeenoX continúa de la siguiente manera

```
READ_MESH square-$2-$3-$4.msh DIMENSIONS 2
PROBLEM neutron_diffusion GROUPS 2

DEFAULT_ARGUMENT_VALUE 1 dirichlet # BCs = dirichlet/neumann
DEFAULT_ARGUMENT_VALUE 2 tri3      # shape = tri3/tri6/quad4/quad8/quad9
DEFAULT_ARGUMENT_VALUE 3 struct    # algorithm = struct/frontal/delaunay
DEFAULT_ARGUMENT_VALUE 4 8         # refinement factor = 1/2/3/4...
DEFAULT_ARGUMENT_VALUE 5 0         # write vtk? = 0/1

# read the results of the symbolic derivatives
INCLUDE neutron-square-s.fee

# set the BCs (depending on $1)
INCLUDE neutron-square-bc-$1.fee

SOLVE_PROBLEM   # this line should be self-explanatory
```

para terminar calculando el error e_2 (y el error e_∞ que ni siquiera discutimos por falta de espacio-tiempo) en forma similar al caso del conejo.

Los dos archivos con condiciones de contorno son el ya conocido 100% Dirichlet:

```
BC left   phi1=phi1_mms(x,y)    phi2=phi2_mms(x,y)
BC right  phi1=phi1_mms(x,y)    phi2=phi2_mms(x,y)
BC bottom phi1=phi1_mms(x,y)    phi2=phi2_mms(x,y)
BC top    phi1=phi1_mms(x,y)    phi2=phi2_mms(x,y)
```

y el nuevo caso 50% Dirichlet 50% Neumann, donde en los lados `bottom` y `right` ponemos el producto interno de la corriente $\mathbf{J}_g(\mathbf{x})$ con la normal exterior $\hat{\mathbf{n}}(\mathbf{x})$:

```
BC left   phi1=phi1_mms(x,y)    phi2=phi2_mms(x,y)
BC top    phi1=phi1_mms(x,y)    phi2=phi2_mms(x,y)
BC bottom J1=+Jy1_mms(x,y)      J2=+Jy2_mms(x,y)
BC right  J1=-Jx1_mms(x,y)      J2=-Jx2_mms(x,y)
```

Como ahora nuestra geometría es más sencilla podemos utilizar algoritmos de mallado estructurados. Incluso podemos estudiar qué sucede si usamos triángulos y cuadrángulos, de primer y segundo orden y completos (quad8) o incompletos (quad9), como someramente ilustramos en la figura 5.52. Con los dos tipos de condiciones de contorno, que podrían ser muchas más combinaciones de porcentajes de Dirichlet y Neumann. ¡Bienvenida la explosión combinatoria!

El resto del trabajo consiste en

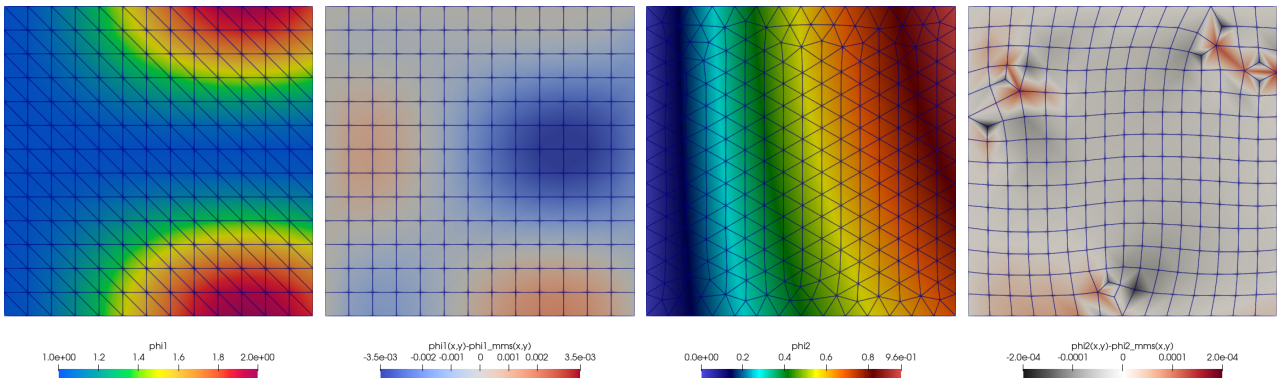
1. ejecutar el script de Bash,
2. dejar que Maxima haga la manipulación simbólica, y
3. obtener con FeenoX los resultados de la figura 5.53.

La siguiente mímica de terminal ilustra la ejecución a través del script `run.sh`:

```
$ ./run.sh
# manufactured solution (input)
phi1_mms(x,y) := 1 + sin(2*x)^2 * cos(3*y)^2;
phi2_mms(x,y) := (1-0.5*tanh(-y))*log(1+x);
D1(x,y) := 1 + 0.1*(x - 0.5*y);
Sigma_a1(x,y) := 1e-3*(1 + log(1+x) - 0.5*y^3);
Sigma_s1_2(x,y) := 1e-3*(1 - x + sqrt(0.5*y));
D2(x,y) := 1;
Sigma_a2(x,y) := 1e-3;
Sigma_s2_1(x,y) := 0;

# source terms (output)
S1(x,y) = (-18*sin(2*x)^2*(0.1*(x-0.5*y)+1)*sin(3*y)^2)-0.3*sin(2*x)^2*cos(3*y)*sin(3*y) ↩
    +0.001*((-0.5*y^3)+log(x+1)+1)*(sin(2*x)^2*cos(3*y)^2+1)+26*sin(2*x)^2*(0.1*(x-0.5*y)+1)* ↩
    cos(3*y)^2-8*cos(2*x)^2*(0.1*(x-0.5*y)+1)*cos(3*y)^2-0.4*cos(2*x)*sin(2*x)*cos(3*y)^2
S2(x,y) = (-0.001*(0.7071067811865476*sqrt(y)-x+1)*(sin(2*x)^2*cos(3*y)^2+1))+1.0*log(x+1)*sech( ↩
    y)^2*tanh(y)+0.001*log(x+1)*(0.5*tanh(y)+1)+(0.5*tanh(y)+1)/(x+1)^2
J1x(x,y) = 4*cos(2*x)*sin(2*x)*((-0.1*(x-0.5*y))-1)*cos(3*y)^2
J1y(x,y) = -6*sin(2*x)^2*((-0.1*(x-0.5*y))-1)*cos(3*y)*sin(3*y)
J2x(x,y) = -(0.5*tanh(y)+1)/(x+1)
J2y(x,y) = -0.5*log(x+1)*sech(y)^2
neutron_square_dirichlet_tri3_struct
-----------------------------------------------------------
-1.732868       -1.940256       -2.857245       4       32      25      68.38   0.85
-1.956012       -2.294614       -3.245481       6       50      36      68.60   1.48
-2.426015       -3.135938       -4.117575       8       128     81      68.32   1.11
[...]
-2.079442       -6.417214       -7.519575       8       64      225     68.92   1.04
-2.302585       -6.909677       -8.086373       10      100     341     67.30   0.68
neutron_square_neumann_quad8_frontal
-----------------------------------------------------------
-1.545521       -4.303977       -5.509111       4       22      83      66.53   1.30
-1.903331       -5.471960       -6.794112       6       45      160     70.89   0.92
-2.191013       -6.055953       -7.497523       8       80      273     69.48   1.55
-2.393746       -6.834924       -8.197128       10      120     401     69.51   1.19
$
```

Observación. La expresión para la componente y de la corriente de grupo 2 involucra una secante hiperbólica—que proviene de la derivada de una tangente hiperbólica—que FeenoX puede evaluar

sin inconvenientes como una de las funciones internas, tales como `log()`, `exp()`, `sin()`, `cos()`, `atan()`, `atan2()`, `random()`, etc.



(a) ϕ_1 tri3 struct (b) $\Delta\phi_1$ quad4 struct (c) ϕ_2 tri3 unstruct (d) $\Delta\phi_2$ quad4 unstruct

Figura 5.52.: Flujos y diferencias en el cuadrado con cuatro tipos de mallas para el caso con condiciones de Neumann.

5.10. PHWR de siete canales y tres barras de control inclinadas

> **TL;DR**: Mallas no estructuradas, dependencias espaciales no triviales, escalabilidad en paralelo. Si el problema no "entra" en una computadora, lo podemos repartir entre varias.

En esta última sección del capítulo resolvemos un problema 100% inventado, desde la geometría (figura 5.54) hasta las secciones eficaces. La geometría es adimensional "tipo" PHWR con siete canales verticales dentro de un tanque moderador y tres barras de control inclinadas.

Observación. La tesis de doctorado [45] trata sobre el modelado de barras de control inclinadas, incluyendo cálculo a nivel de celda y a nivel de núcleo. En el nivel de celda se tiene en cuenta la geometría continua para calcular secciones eficaces macroscópicas. Pero luego estas secciones eficaces son usadas a nivel de núcleo en una malla estructurada obteniendo los mismos dibujos tipo "Lego" que mostramos en la figura 1.7 de la sección 1.2. Este problema propone tener en cuenta la inclinación de las barras en el cálculo de núcleo. Queda como trabajo futuro el análisis de la forma correcta de generar las secciones eficaces con cálculo a nivel de celda para garantizar la consistencia del esquema de cálculo multi-escala (sección 2.5).

Luego, en esta sección las secciones eficaces son, tal como la geometría, 100% inventadas. Más aún, para el moderador, el invento incluye una dependencia con un perfil (lineal) de la temperatura del moderador en función de la coordenada vertical:

```
Tmod0 = 100
Tmod(z) = 100 + (200-100)*(z/400)
FUNCTION sigmat_mod(T) INTERPOLATION linear DATA {
90  0.108
100 0.112
110 0.116
130 0.118
```

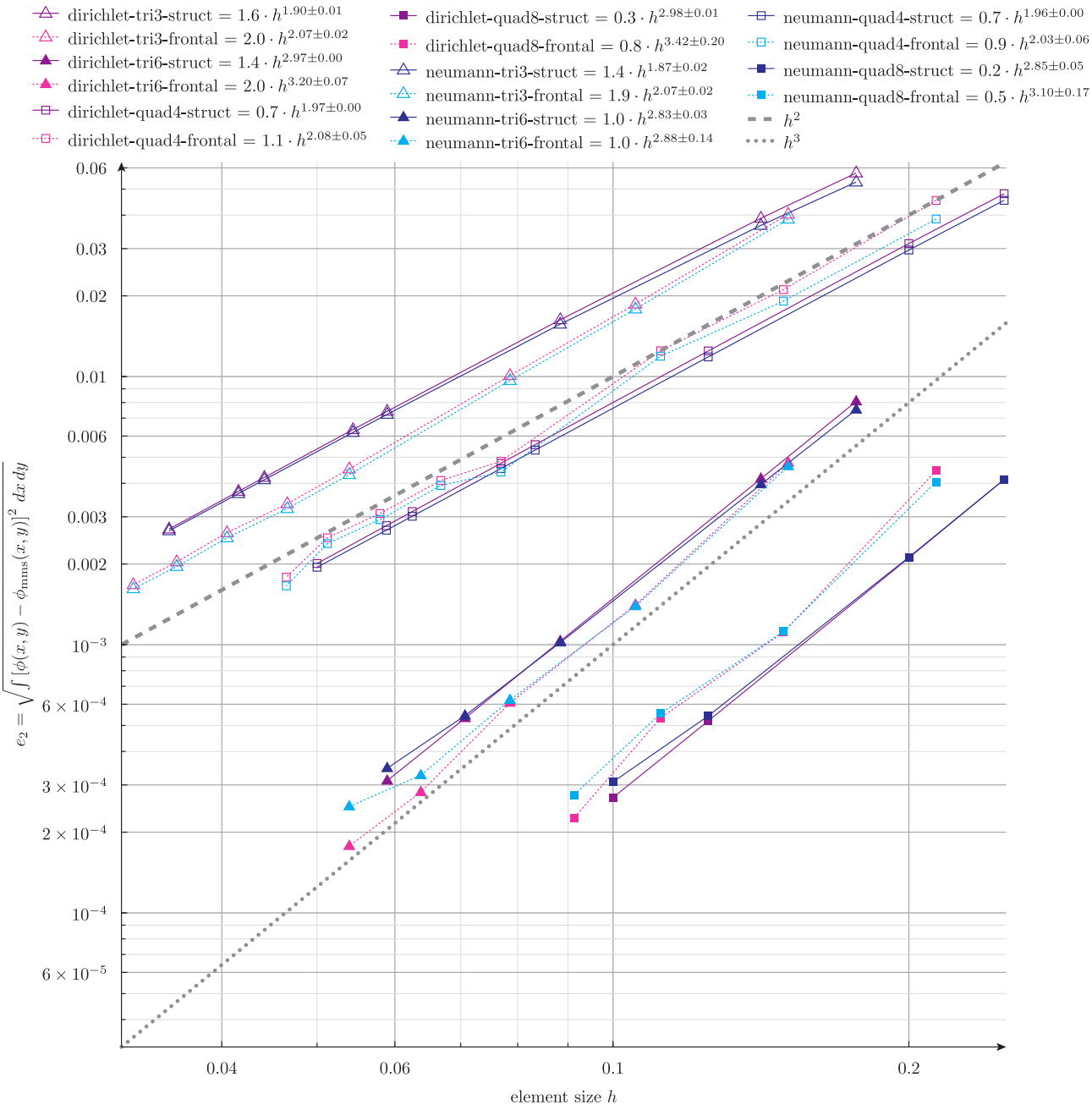


Figura 5.53.: Error e_2 vs. tamaño del elemento h para el cuadrado a dos grupos

```
180 0.120
200 0.122 }

MATERIAL moderator {
  Sigma_t1=0.068
  Sigma_t2=sigmat_mod(Tmod(z))
  Sigma_s1.1=0.06+1e-5*(Tmod(z)-Tmod0)
  Sigma_s1.2=0.002
  Sigma_s2.1=0
  Sigma_s2.2=sigmat_mod(Tmod(z))-0.005
  nuSigma_f1=0
  nuSigma_f2=0
}
```

Observación. De acuerdo con la idea de evitar complejidad innecesaria, las secciones eficaces de los canales combustibles son uniformes. Pero en una aplicación real, éstas deberían depender de...

- el quemado
- la concentración de venenos
- la temperatura del combustible
- la temperatura del refrigerante
- la densidad del refrigerante

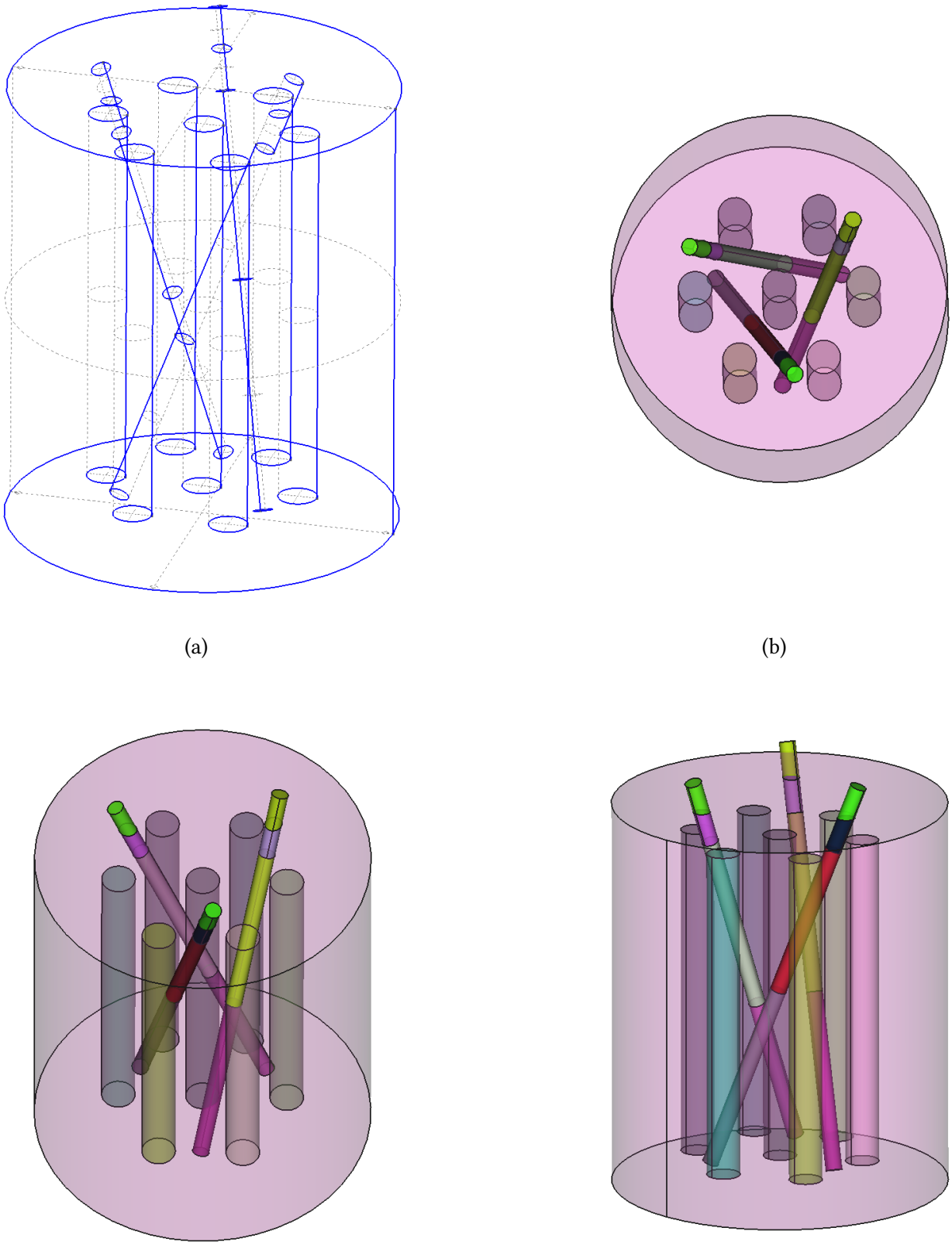
Todas estas dependencias se pueden dar en forma similar a la dependencia de las XS del moderador con una mezcla de expresiones algebraicas (sección 4.2) e interpolación de funciones dadas por puntos de una o más variables (sección 4.3). Incluso se pueden diseñar esquemas de acople con códigos externos e intercambiar información a través de memoria compartida [75], [87] o de mensajes MPI.

5.10.1. Difusión con elementos de segundo orden

Resolvemos primero las ecuaciones de difusión a dos grupos sobre una malla con elementos curvos tipo tet10 (figura 5.55). El archivo de entrada de FeenoX, una vez que separamos la secciones eficaces (y hemos puesto toda la complejidad en la geometría y en la malla) es extremadamente sencillo:

```
PROBLEM neutron_diffusion 3D GROUPS 2 PRECONDITIONER mumps
READ_MESH phwr2.msh INTEGRATION reduced
INCLUDE xs.fee
SOLVE_PROBLEM

WRITE_RESULTS FORMAT vtk

mem_global = mpi_memory_global()
PRINTF     "size = %g\ttime = %.1f s\t memory = %.1f Gb" total_dofs wall_time() mem_global
PRINTF_ALL "local memory = %.1f Gb" mpi_memory_local()
```

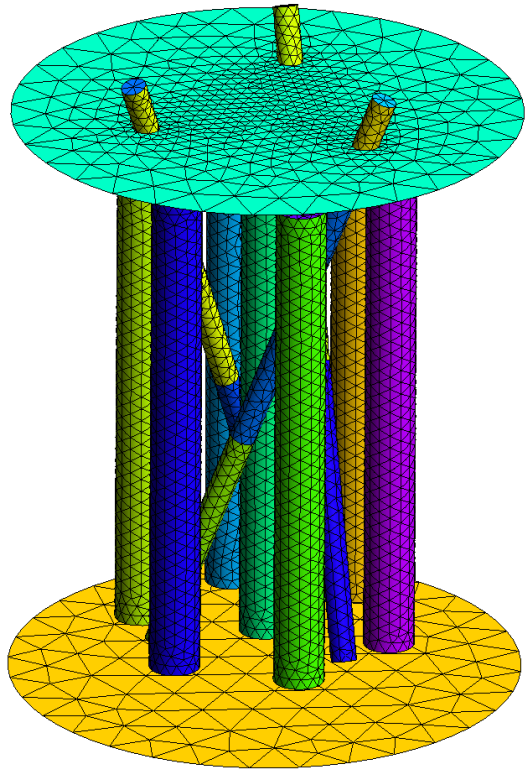
A dos grupos, la malla de segundo orden resulta en poco más de 250 mil grados de libertad. Veamos cómo escala FeenoX en términos de tiempo y memoria al resolver este problema con diferente cantidad de procesadores en una misma computadora:



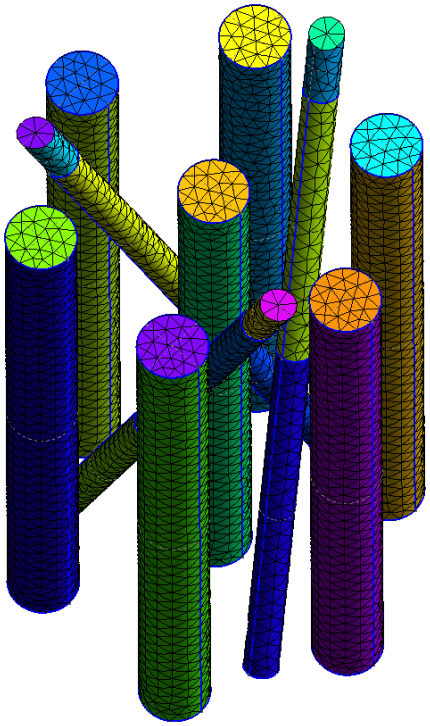
(a) (b)

(c) (d)

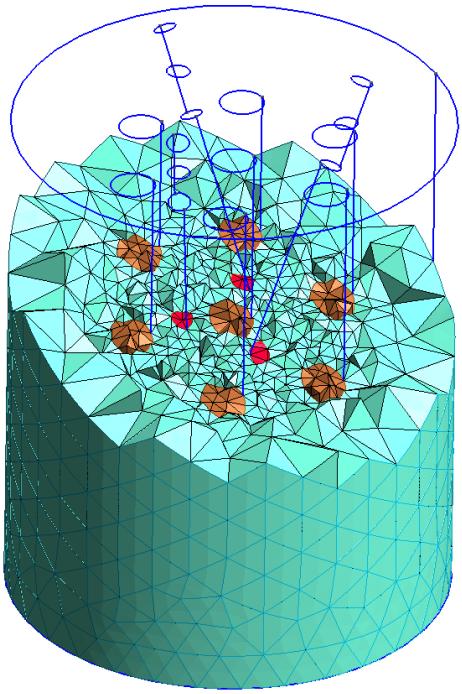
Figura 5.54.: Geometría de un PHWR inventado con 7 canales verticales y 3 barras de control



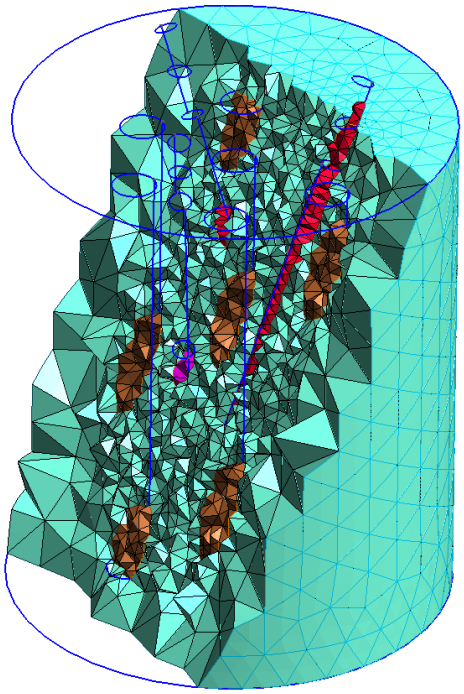
(a)



(b)



(c)



(d)

Figura 5.55.: Malla con elementos curvos tet10 para el PHWR inventado

```
$ for i in 1 2 4 8 12; do mpirun -n $i feenox phwr-dif.fee; done
size = 256964   time = 94.6 s    memory = 6.5 Gb
[0/1 tux] local memory = 6.5 Gb
size = 256964   time = 61.5 s    memory = 7.9 Gb
[0/2 tux] local memory = 4.2 Gb
[1/2 tux] local memory = 3.7 Gb
size = 256964   time = 49.7 s    memory = 9.5 Gb
[0/4 tux] local memory = 2.3 Gb
[1/4 tux] local memory = 2.4 Gb
[2/4 tux] local memory = 2.4 Gb
[3/4 tux] local memory = 2.4 Gb
size = 256964   time = 46.2 s    memory = 12.9 Gb
[0/8 tux] local memory = 1.6 Gb
[1/8 tux] local memory = 1.6 Gb
[2/8 tux] local memory = 1.9 Gb
[3/8 tux] local memory = 1.5 Gb
[4/8 tux] local memory = 1.6 Gb
[5/8 tux] local memory = 1.5 Gb
[6/8 tux] local memory = 1.5 Gb
[7/8 tux] local memory = 1.7 Gb
size = 256964   time = 48.8 s    memory = 15.1 Gb
[0/12 tux] local memory = 1.3 Gb
[1/12 tux] local memory = 1.2 Gb
[2/12 tux] local memory = 1.1 Gb
[3/12 tux] local memory = 1.4 Gb
[4/12 tux] local memory = 1.5 Gb
[5/12 tux] local memory = 1.2 Gb
[6/12 tux] local memory = 1.1 Gb
[7/12 tux] local memory = 1.1 Gb
[8/12 tux] local memory = 1.2 Gb
[9/12 tux] local memory = 1.0 Gb
[10/12 tux] local memory = 1.6 Gb
[11/12 tux] local memory = 1.4 Gb
$
```

Si bien el tiempo de pared disminuye, no lo hace tanto como debería ya que todavía hay mucho lugar para optimización en FeenoX, especialmente en paralelización por MPI. Pero podemos observar que el comportamiento es esencialmente el esperado. Más importante aún es el comportamiento de la memoria: a medida que usamos más procesos (o "ranks" en terminología de MPI), la memoria requerida en cada uno disminuye sensiblemente. Esto implica que FeenoX puede—en principio—resolver problemas arbitrariamente grandes si se dispone de suficientes computadoras que puedan ser interconectadas por MPI, que era una de las premisas de esta tesis.[11] La figura 5.56 muestra la distribución de flujos rápido y térmico resultantes.

5.10.2. Ordenadas discretas con elementos de primer orden

Resolvamos ahora el mismo problema pero con ordenadas discretas. Comenzamos por S_2, que involucra ocho direcciones por cada grupo de energías. Para tener un tamaño de problema comparable

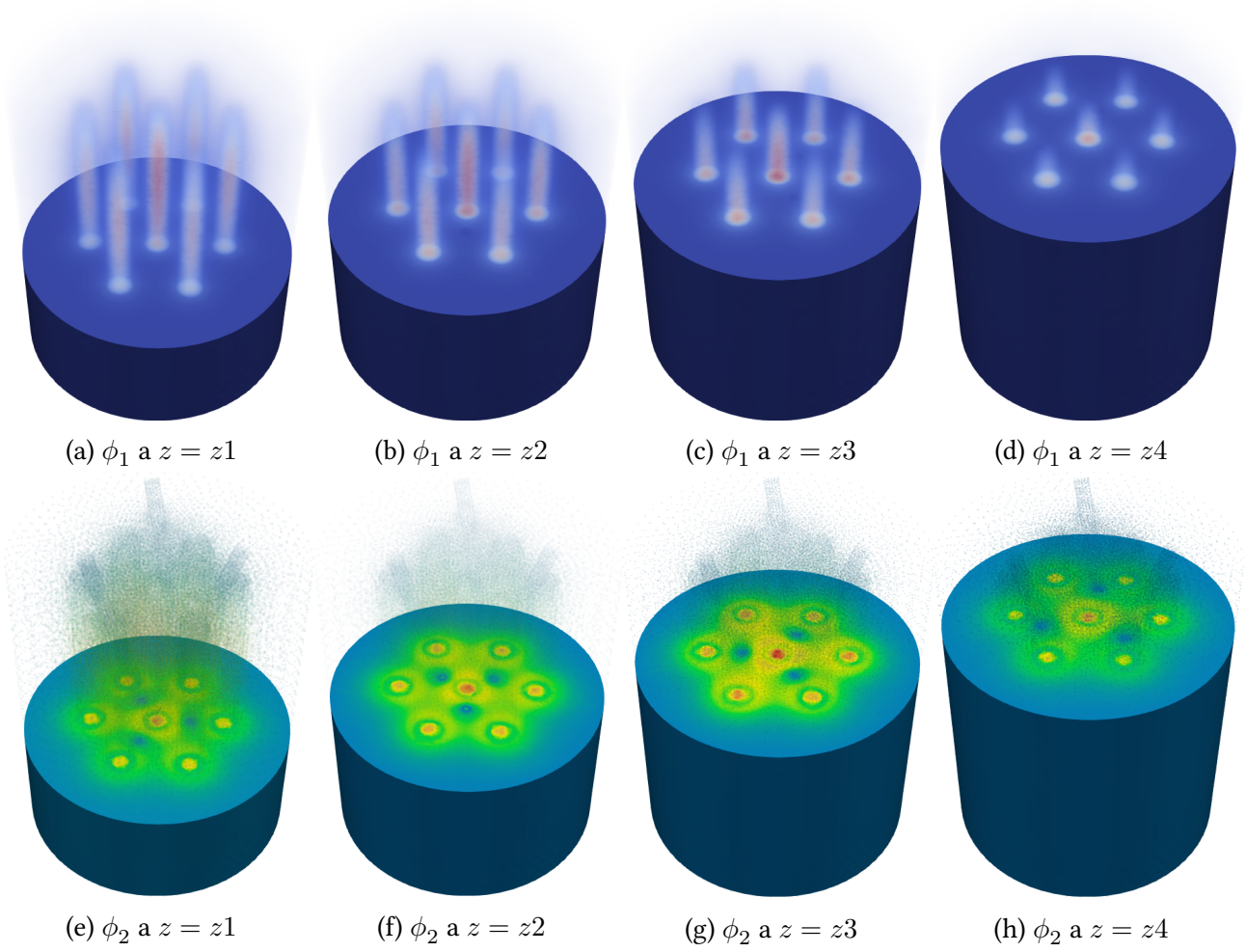
[11] Quedan como trabajos futuros el análisis de convergencia de otros pre-condicionadores y el estudio de escalabilidad en paralelo de problemas tipo S_N (sección 6.1}).

utilizamos tetraedros de primer orden. Estudiemos cómo cambia el tiempo de pared y la memoria con 1, 2, 4 y 8 procesos MPI:

```
$ for i in 1 2 4 8; do mpirun -n $i feenox phwr-s2.fee; done
size = 257920   time = 409.7 s   memory = 20.1 Gb
[0/1 tux] local memory = 20.1 Gb
size = 257920   time = 286.3 s   memory = 25.5 Gb
[0/2 tux] local memory = 11.5 Gb
[1/2 tux] local memory = 14.1 Gb
size = 257920   time = 289.3 s   memory = 29.5 Gb
[0/4 tux] local memory = 7.6 Gb
[1/4 tux] local memory = 6.8 Gb
[2/4 tux] local memory = 7.2 Gb
[3/4 tux] local memory = 8.0 Gb
size = 257920   time = 182.7 s   memory = 33.9 Gb
[0/8 tux] local memory = 4.5 Gb
[1/8 tux] local memory = 4.4 Gb
[2/8 tux] local memory = 4.4 Gb
[3/8 tux] local memory = 4.6 Gb
[4/8 tux] local memory = 4.3 Gb
[5/8 tux] local memory = 4.1 Gb
[6/8 tux] local memory = 3.7 Gb
[7/8 tux] local memory = 4.0 Gb
$
```

Resolver un problema formulado en S_N es computacionalmente mucho más demandante porque las matrices resultantes no son simétricas y tienen una estructura compleja. Los requerimientos de memoria y CPU son mayores que para difusión. Incluso la escala de paralelización, aún cuando debemos notar nuevamente que hay mucho terreno para mejorar en FeenoX, es peor que en la sección anterior para un tamaño de problema similar. El esfuerzo necesario es más marcado mientras mayor sea N. De hecho para una malla más gruesa todavía, dando lugar a un tamaño de problema menor, obtenemos:

```
$ mpiexec -n 1 feenox phwr-s4.fee
size = 159168   time = 390.1 s   memory = 20.0 Gb
[0/1 tux] local memory = 20.0 Gb
$ mpiexec -n 2 feenox phwr-s4.fee
size = 159168   time = 297.6 s   memory = 25.0 Gb
[0/2 tux] local memory = 12.5 Gb
[1/2 tux] local memory = 12.5 Gb
$ mpiexec -n 4 feenox phwr-s4.fee
size = 159168   time = 276.7 s   memory = 27.5 Gb
[0/4 tux] local memory = 7.7 Gb
[1/4 tux] local memory = 6.9 Gb
[2/4 tux] local memory = 6.2 Gb
[3/4 tux] local memory = 6.6 Gb
$ mpiexec -n 8 feenox phwr-s4.fee
size = 159168   time = 153.3 s   memory = 33.7 Gb
[0/8 tux] local memory = 5.0 Gb
[1/8 tux] local memory = 4.7 Gb
[2/8 tux] local memory = 3.2 Gb
[3/8 tux] local memory = 3.5 Gb
[4/8 tux] local memory = 3.5 Gb
[5/8 tux] local memory = 4.8 Gb
```



(a) ϕ_1 a $z = z1$ (b) ϕ_1 a $z = z2$ (c) ϕ_1 a $z = z3$ (d) ϕ_1 a $z = z4$

(e) ϕ_2 a $z = z1$ (f) ϕ_2 a $z = z2$ (g) ϕ_2 a $z = z3$ (h) ϕ_2 a $z = z4$

Figura 5.56.: Flujos escalares rápido ϕ_1 y térmico ϕ_2 calculados con difusión. Todos los combustibles tienen el mismo quemado.

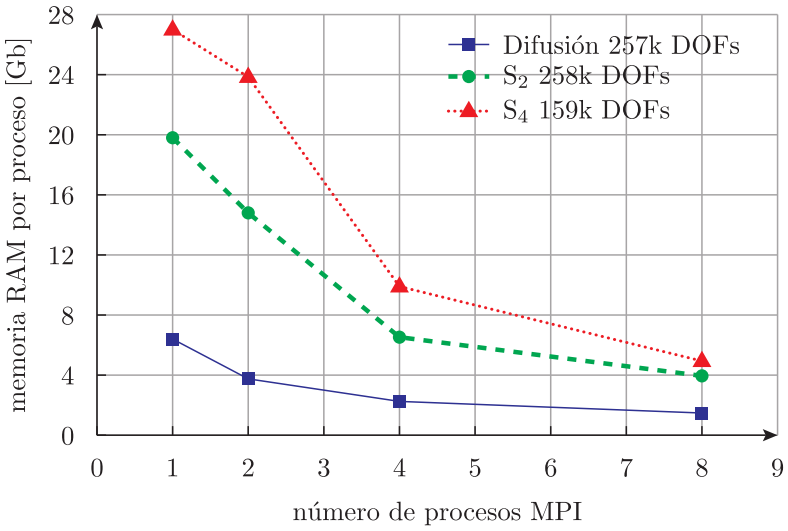


Figura 5.57.: Disminución de la memoria por proceso MPI

Formulación	DOFs	Problema	Build	Solve	Total	Mem.
Difusión	257k	KSP	3.5 s	5.4 s	10.6 s	0.6 Gb
		EPS	6.9 s	66.3 s	74.7 s	6.4 Gb
S_2	258k	KSP	30.4 s	228.1 s	260.0 s	17.7 Gb
		EPS	63.2 s	368.6 s	432.9 s	19.8 Gb
S_4	159k	KSP	64.7 s	449.1 s	514.6 s	23.3 Gb
		EPS	138.9 s	598.8 s	738.3 s	27.0 Gb

(a)

Tabla 5.8.: Tiempos necesarios para construir y resolver diferentes formulaciones para casos con fuentes (KSP) o de criticidad (EPS)

```
[6/8 tux] local memory = 4.8 Gb
[7/8 tux] local memory = 4.3 Gb
$
```

Esto es, para el mismo número de grados de libertad totales el tiempo y memoria necesario para resolver el problema con S_4 aumenta. De todas maneras, lo que sí sigue siendo cierto, como mostramos en la figura 5.57, es que a medida que aumentamos la cantidad de procesos de MPI la memoria local disminuye.

Para finalizar, debemos notar que al resolver problemas de criticidad lo que FeenoX hace es transformar la formulación numérica desarrollada en el capítulo 3 en un problema de auto-valores y auto-vectores generalizado como explicamos en la sección 3.5.3. Para resolver este tipo de problemas se necesita un solver lineal que pueda "invertir"[12] la matriz de fisiones. Los algoritmos para resolver problemas de autovalores provistos en la biblioteca SLEPc funcionan significativamente mejor si este solver lineal es directo. Es conocido que los solvers directos son robustos pero no son escalables. Por lo tanto, los problemas resueltos con FeenoX (usando las opciones por defecto) suelen ser robustos pero no escalan bien (de hecho en la sección 5.3.5 hemos resuelto un problema de criticidad con un solver lineal usando opciones en la línea de comandos). Es por eso también que los problemas sin fuentes independientes son más intensivos computacionalmente que los problemas con fuentes, que pueden ser resueltos como un sistema de ecuaciones lineales (o eventualmente no lineales con un esquema tipo Newton-Raphson).

En efecto, como vemos en la tabla 5.8, en el caso de difusión con fuentes independientes, la matriz de rigidez es simétrica y el operador es elíptico. Esto hace que sea muy eficiente usar un precondicionador geométrico-algebraico multi-grilla (GAMG) combinado con un solver de Krylov tipo gradientes conjugados, tanto en términos de CPU como de memoria. Justamente esa combinación es el *default* para problemas tipo `neutron_diffusion` en FeenoX. Por otro lado, al resolver `neutron_sn`, aún para problemas con fuente se necesita un solver directo ya que de otra manera la convergencia es muy lenta con un impacto directo en la cantidad de memoria necesaria.

Observación. En la sección 5.9 hemos verificado solamente la primera fila de la tabla 5.8.

[12]En el sentido de resolver un problema lineal, no de calcular explícitamente la inversa densa de una matriz rala.

5.11. Bonus track: cinética puntual*

> **TL;DR**: Ilustración del hecho de que FeenoX puede resolver ecuaciones diferenciales ordinarias además de en derivadas parciales.

Además de ecuaciones en derivadas parciales, FeenoX puede resolver sistemas de ecuaciones diferenciales ordinarias y de ecuaciones algebraicas-diferenciales. En esta sección extra ilustramos rápidamente las funcionalidades, aplicadas a las ecuaciones de cinética puntual de reactores. Todos los casos usan los siguientes parámetros cinéticos, definido en un archivo `parameters.fee` e incluido en los inputs de cada una de las secciones que siguen:

```
nprec = 6    # seis grupos de precursores
VECTOR c[nprec]
VECTOR lambda[nprec] DATA 0.0124   0.0305   0.111    0.301    1.14     3.01
VECTOR beta[nprec]   DATA 0.000215 0.001424 0.001274 0.002568 0.000748 0.000273
Beta = vecsum(beta)
Lambda = 40e-6
```

5.11.1. Cinética puntual directa con reactividad vs. tiempo

Este primer ejemplo resuelve cinética puntual con una reactividad $\rho(t)$ dada por una "tabla",[13] es decir, una función de un único argumento (el tiempo t) definida por pares de puntos $[t, \rho(t)]$ e interpolada linealmente:

```
INCLUDE parameters.fee   # parámetros cinéticos
PHASE_SPACE phi c rho    # espacio de fases
end_time = 100           # tiempo final

rho_0 = 0                # condiciones iniciales
phi_0 = 1
c_0[i] = phi_0 * beta[i]/(Lambda*lambda[i])

# "tabla" de reactividad vs. tiempo en pcm
FUNCTION react(t) DATA {   0    0
                           5    0
                           10  10
                           30  10
                           35   0
                           100  0   }

# sistema de DAEs
rho = 1e-5*react(t)
phi_dot = (rho-Beta)/Lambda * phi + vecdot(lambda, c)
c_dot[i] = beta[i]/Lambda * phi - lambda[i]*c[i]

PRINT t phi rho    # salida: phi y rho vs. tiempo
```

La figura 5.58 muestra el nivel de flujo $\phi(t)$ y la reactividad $\rho(t)$ tal como la utilizó FeenoX para resolver el sistema dinámico definido con la palabra clave `PHASE_SPACE` (que es un sustantivo).

[13] Recordar que uno de los puntos centrales de la filosofía de diseño de FeenoX es evitar el ambiguo y anacrónico concepto de "tabla" en favor de "función definida por puntos".

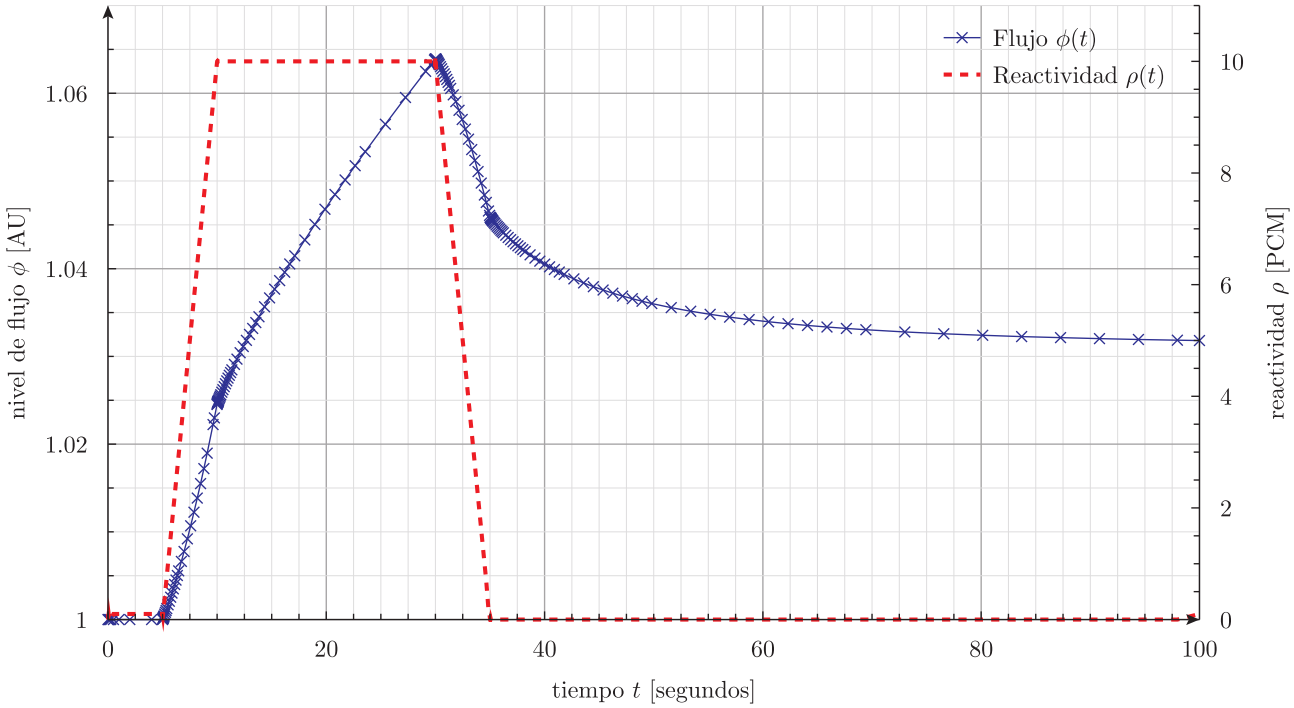


Figura 5.58.: Flujo y reactividad directa

5.11.2. Cinética inversa

Ahora tomamos la salida $\phi(t)$ del caso anterior y resolvemos cinética inversa de dos maneras diferentes:

1. Con la fórmula integral de la literatura clásica [15]

```
INCLUDE parameters.fee
FUNCTION flux(t) FILE flux.dat

# definimos una función de flujo que permite tiempos negativos
flux_a = vec_flux_t[1]
flux_b = vec_flux_t[vecsize(vec_flux)]
phi(t) = if(t<flux_a, flux(flux_a), flux(t))

# calculamos la reactividad con la fórmula integral
VAR t'
rho(t) := { Lambda * derivative(log(phi(t')),t',t) +
  Beta * ( 1 - 1/phi(t) *
   integral(phi(t-t') * sum((lambda[i]*beta[i]/Beta)*exp(-lambda[i]*t'), i, 1, nprec),
       t', 0, 1e4) ) }

PRINT_FUNCTION rho MIN 0 MAX 50 STEP 0.1
```

2. Resolviendo el mismo sistema de DAEs pero leyendo $\phi(t)$ en lugar de $\rho(t)$

```
INCLUDE parameters.fee
PHASE_SPACE phi c rho

end_time = 50
dae_rtol = 1e-7
```

```
rho_0 = 0
phi_0 = 1
c_0[i] = phi_0 * beta[i]/(Lambda*lambda[i])

FUNCTION flux(t) FILE flux.dat

phi = flux(t)
phi_dot = (rho-Beta)/Lambda * phi + vecdot(lambda, c)
c_dot[i] = beta[i]/Lambda * phi - lambda[i]*c[i]

PRINT t phi rho
```

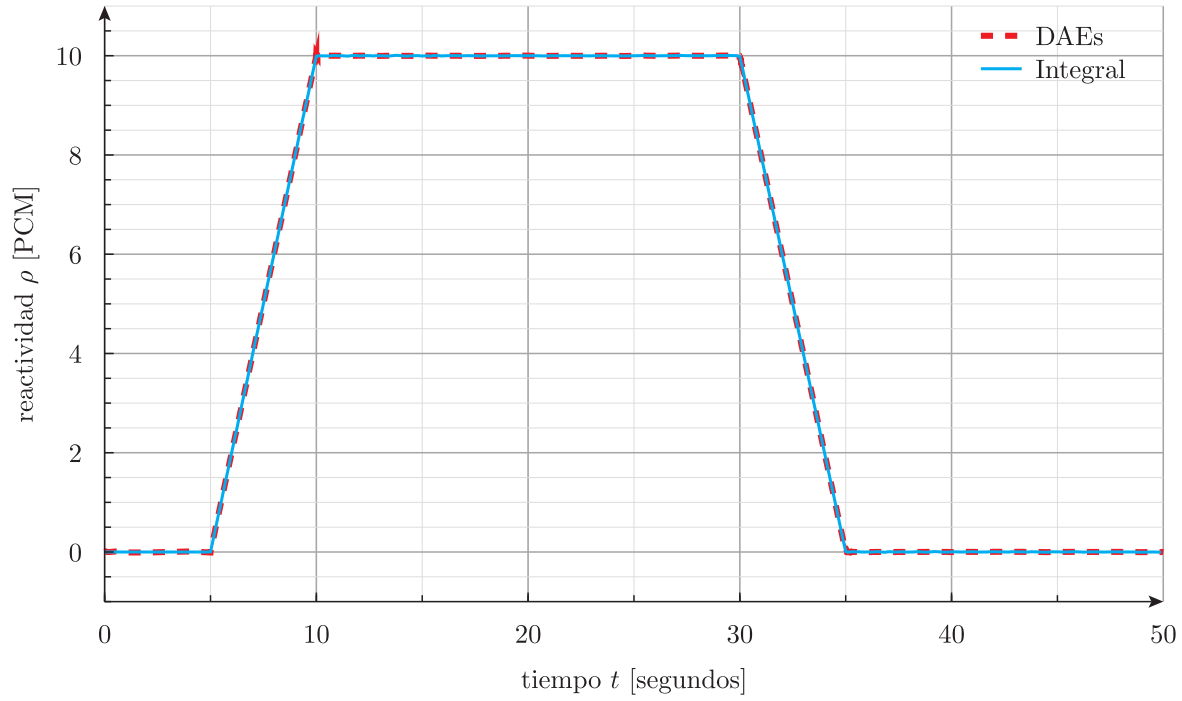
Obtenemos entonces la figura 5.59. El caso 2 es "adaptativo" en el sentido de que dependiendo del error tolerado y de las derivadas temporales de las variables del espacio de las fases en función de t, el esfuerzo computacional se adapta automáticamente a través del paso de tiempo Δt con el que se resuelve el sistema DAE. Por defecto, el método es Adams-Bashforth de orden variable (implementado por la biblioteca SUNDIALS [19]), donde justamente el orden de integración se ajusta dinámicamente también dependiendo del error.

5.11.3. Control de inestabilidades de xenón

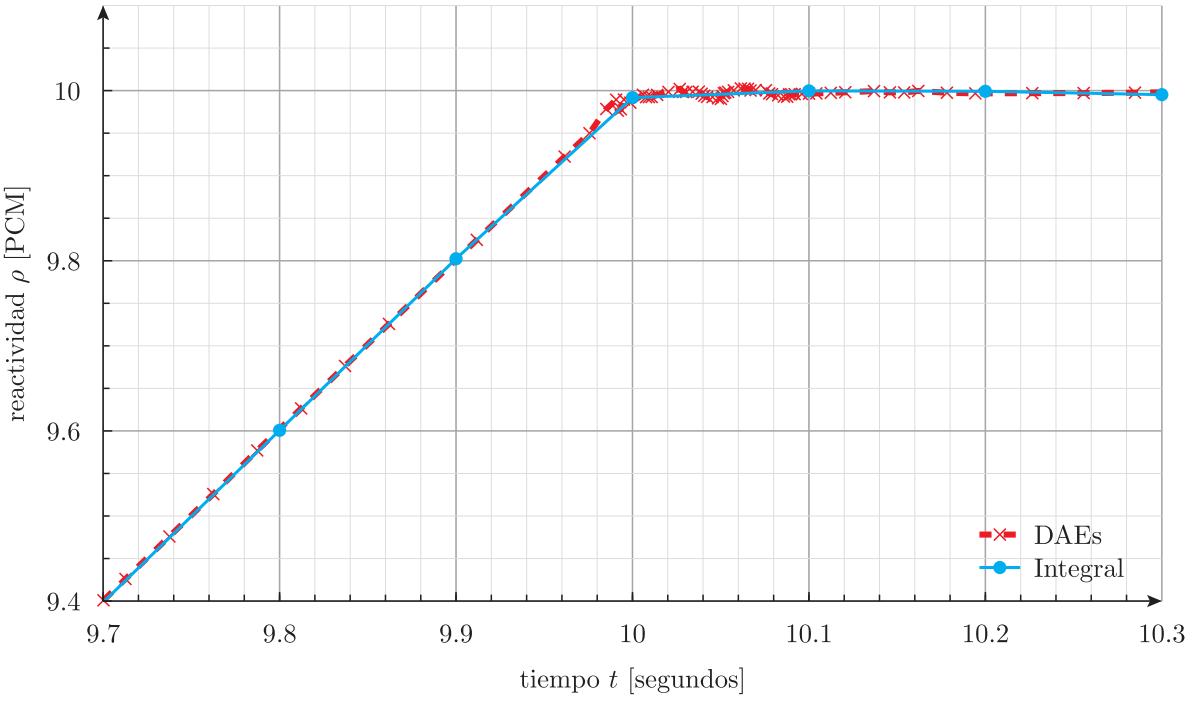
Ahora introducimos un poco más de complejidad. A las ecuaciones de cinética puntual le agregamos cinética de xenón 135. Como el sistema dinámico resultante es Lyapunov-inestable[14] ante cambios de flujo, la reactividad es ahora una función de la posición de una barra de control ficticia cuya importancia está dada por una interpolación tipo Steffen de su posición adimensional z. Una lógica de control PI (con una banda muerta del 0.3%) "mueve" dicha barra de control de forma tal de forzar al reactor a bajar la potencia del 100% al 80% en mil segundos, mantenerse durante tres mil segundos a esa potencia y volver al 100% en cinco mil:

```
INCLUDE parameters.fee
FUNCTION setpoint(t) DATA {
0     1
1000  1
2000  0.8
5000  0.8
10000 1
20000 1 }

end_time = vecmax(vec_setpoint_t)  # tiempo final = último tiempo de setpoint(t)
max_dt = 1                         # no dejamos que dt aumente demasiado

# importancia de la barra de control como función de la inserción
FUNCTION rodworth(z) INTERPOLATION akima DATA {
0     2.155529e+01*1e-5*10
0.2   6.337352e+00*1e-5*10
0.4  -3.253021e+01*1e-5*10
0.6  -7.418505e+01*1e-5*10
0.8  -1.103352e+02*1e-5*10
1    -1.285819e+02*1e-5*10  }
```

[14] Para observar "oscilaciones de xenón" es necesario a. un lazo de control a lazo cerrado y b. cinética espacial o multipuntual.



(a) $t \in [0, 100]$



(b) $t \in [9.75, 10.25]$

Figura 5.59.: Reactividad calculada mediante cinética inversa de dos maneras diferentes

```
# constantes para el xenón
gammaX = 1.4563E10       # xenon-135 direct fission yield
gammaI = 1.629235E11     # iodine-135 direction fission yield
GammaX = -3.724869E-17   # xenon-135 reactivity coefficiente
lambdaX = 2.09607E-05    # xenon-135 decay constant
lambdaI = 2.83097E-05    # iodine-135 decay constant
sigmaX = 2.203206E-04    # microscopic XS of neutron absorption for Xe-134

PHASE_SPACE rho phi c I X
INITIAL_CONDITIONS_MODE FROM_VARIABLES

z_0 = 0.5                # estado estacionario
phi_0 = 1
c_0[i] = phi_0 * beta[i]/(Lambda*lambda[i])
I_0 = gammaI*phi_0/lambdaI
X_0 = (gammaX + gammaI)/(lambdaX + sigmaX*phi_0) * phi_0
rho_bias_0 = -rodworth(z_0) - GammaX*X_0

# --- DAEs ------------------------------
rho = rho_bias + rodworth(z) + GammaX*X
phi_dot = (rho-Beta)/Lambda * phi + vecdot(lambda, c)
c_dot[i] = beta[i]/Lambda * phi - lambda[i]*c[i]
I_dot = gammaI * phi - lambdaI * I
X_dot = gammaX * phi + lambdaI * I - lambdaX * X - sigmaX * phi * X

# --- sistema de control ----------------
# movemos la barra de control si el error excede una banda muerta del 0.3%
vrod = 1/500     # 1/500-avos de núcleo por segundo
band = 3e-3
error = phi - setpoint(t)
z = z_0 + integral_dt(vrod*((error>(+band))-(error<(-band))))

PRINT t phi z setpoint(t)
```

La figura 5.60 muestra el flujo y la posición de la barra de control. Se puede observar que la dinámica no es trivial, pero puede ser modelada en forma relativamente sencilla con FeenoX.

5.11.4. Mapas de diseño

Finalizamos recuperando unos resultados derivados de mi tesis de maestría [61] publicados en 2010 [71]. Consiste en cinética puntual de un reactor de investigación con retroalimentación termohidráulica por temperatura del refrigerante y del combustible escrita como modelos de capacitancia concentrada[15] cero-dimensionales. El estudio consiste en barrer paramétricamente el espacio de coeficientes de reactividad $[\alpha_c, \alpha_f]$, perturbar el estado del sistema dinámico ($\Delta T_f = 2$ °C) y marcar con un color la potencia luego de un minuto para obtener mapas de estabilidad tipo Lyapunov.

```
nprec = 6    # six precursor groups
VECTOR c[nprec]
VECTOR lambda[nprec] DATA 1.2400E-02 3.0500E-02 1.1100E-01 3.0100E-01 1.1400E+00 3.0100E+00
VECTOR beta[nprec]   DATA 2.4090e-04 1.5987E-03 1.4308E-03 2.8835E-03 8.3950E-04 3.0660E-04
Beta = vecsum(beta)
Lambda = 1.76e-4

IF in_static
```

[15] Del inglés *lumped capacitance*.

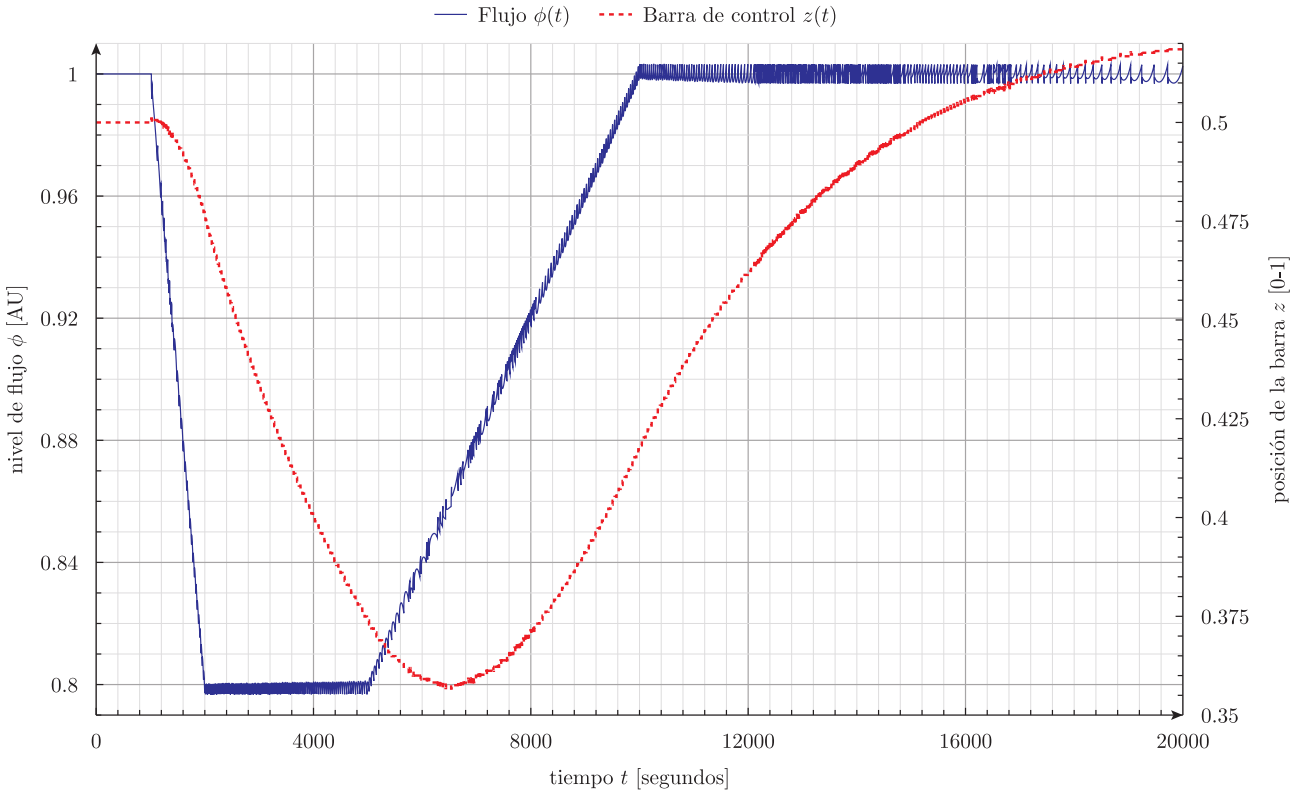


Figura 5.60.: Flujo y posición de la barra de control en un caso con xenón bajo control

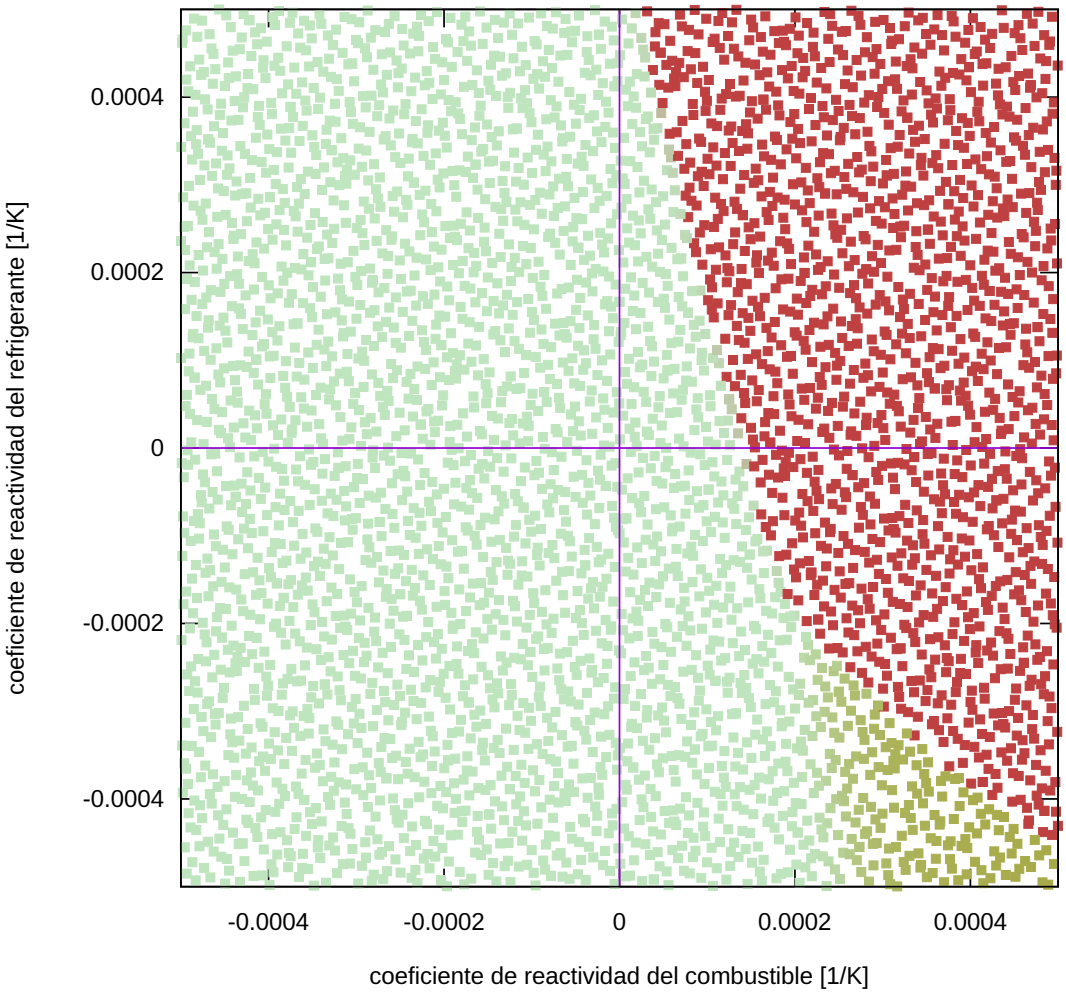
```
 alpha_T_fuel = 100e-5*(qrng2d_reversehalton(1,$1)-0.5)
 alpha_T_cool = 100e-5*(qrng2d_reversehalton(2,$1)-0.5)

 Delta_T_cool = 2
 Delta_T_fuel = 0

 P_star = 18.8e6        # watts
 T_in = 37              # grados C
 hA_core = 1.17e6       # watt/grado
 mc_fuel = 47.7e3       # joule/grado
 mc_cool = 147e3        # joule/grado
 mflow_cool = 520       # kg/seg
 c_cool = 4.18e3 * 147e3/mc_cool       # joule/kg
ENDIF

PHASE_SPACE phi c T_cool T_fuel rho
end_time = 60
dae_rtol = 1e-7

rho_0 = 0
phi_0 = 1
c_0[i] = phi_0 * beta(i)/(Lambda*lambda(i))

T_cool_star = 1/(2*mflow_cool*c_cool) * (P_star+2*mflow_cool*c_cool*T_in)
T_fuel_star = 1/(hA_core) * (P_star + hA_core*T_cool_star)

T_cool_0 = T_cool_star + Delta_T_cool
T_fuel_0 = T_fuel_star + Delta_T_fuel
INITIAL_CONDITIONS_MODE FROM_VARIABLES
```

```
rho = 0
phi_dot = (rho + alpha_T_fuel*(T_fuel-T_fuel_star) + alpha_T_cool*(T_cool-T_cool_star) -
    Beta)/Lambda * phi + vecdot(lambda, c)
c_dot[i] = beta[i]/Lambda * phi - lambda[i]*c[i]
T_fuel_dot = (1.0/(mc_fuel))*(P_star*phi - hA_core*(T_fuel-T_cool))
T_cool_dot = (1.0/(mc_cool))*(hA_core*(T_fuel-T_cool) - 2*mflow_cool*c_cool*(T_cool-T_in))

done = done | (phi > 4)

IF done
 PRINT alpha_T_fuel alpha_T_cool phi
ENDIF
```

Para barrer el espacio de parámetros usamos series de números cuasi-aleatorios [22] de forma tal de poder realizar ejecuciones sucesivas que van llenando densamente dicho espacio como ilustramos en la figura 5.61a:

```
for i in $(seq $1 $2); do  feenox point.fee $i | tee -a point.dat; done
```

```
$ ./point.sh 0 2048
$ ./point.sh 2048 4096
$
```



(a) Estabilidad de Lyapunov utilizando series de números pseudo-aleatorios que van "rellenando" incremental y densamente el espacio de parámetros.

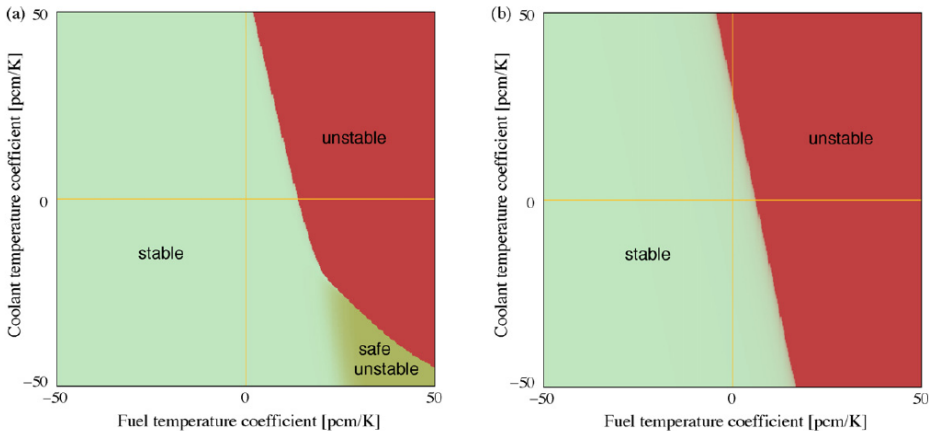


Fig. 6. Stability maps in the $\alpha_{T_{cool}}$ vs. $\alpha_{T_{fuel}}$ parameter plane. The perturbation in the Lyapunov study is a perturbation of 2 ° C in the inlet temperature of the coolant. The perturbation in the BIBO study is a positive 10 pcm external reactivity insertion.

(b) Figuras originales de la referencia [71]

Figura 5.61.: Mapas de estabilidad de cinética puntual con realimentación termohidráulica

6. Conclusiones

> Lejos está de mí la presunción de pretender dictar aquí el método que cada uno debe seguir para dirigir bien su razón. Sólo quiero mostrar el camino que he elegido para conducir la mía; porque los que se erigen en preceptores deben creerse más dotados que aquellos a quienes pretender aleccionar. Entonces ofrezco este escrito como una historia o fábula, algunos de cuyos ejemplos pueden seguirse, en tanto otros pueden ser desechados, tomando cada uno lo que le aproveche. Por eso pienso que este discurso, ha de ser útil para muchos y no hacer daño a nadie.
> *René Descartes, Discurso del método, 1637*

> Leí con incomprensión y fervor estas palabras que con minucioso pincel redactó un hombre de mi sangre:
>
> > Dejo a los varios porvenires (no a todos) mi jardín de senderos que se bifurcan.
>
> Devolví en silencio la hoja.
> *Jorge Luis Borges, El jardín de senderos que se bifurcan, 1941*

Con esta tesis de alguna manera cerramos dos lazos:

a. Uno estrictamente personal que involucra un poco más de quince años abarcando

   - la tesis de grado sobre control de lazos de convección natural caóticos [60]
   - la tesis de maestría sobre inestabilidades no lineales en el problema acoplado termohidráulico-neutrónico [61]
   - este extenso trabajo, medido tanto en tiempo de desarrollo (principalmente entendiendo qué es lo que *no* hay que hacer) como en cantidad de texto.

b. Otro más general en el que agregamos el nivel de núcleo a las dos tesis de doctorado

- "Desarrollo y aplicaciones de nuevas bibliotecas de secciones eficaces neutrónicas para H_2O, D_2O y HDO" de José Ignacio Márquez Damián [37]
- "Metodología de análisis neutrónico de celdas de reactores de agua pesada" de Héctor Lestani [33]

En ella hemos recorrido los tres aspectos en el no tan tradicional y poco académico pero—a veces—útil orden

1. ¿Por qué? (cap. 1)
2. ¿Cómo? (cap. 2 y 3)
3. ¿Qué? (cap. 4 y 5)

La idea del desarrollo se basa en comenzar con un documento ficticio (pero plausible) con un Software Requirements Specification (apéndice A) en el cual un cliente—que podría ser una entidad pública, un laboratorio o una compañía privada—especifica un pliego de condiciones técnicas que debe tener una herramienta computacional para ser comprada o financiada. FeenoX aparece como una "oferta" a dicho pliego, con un Software Design Specification (apéndice B). Este enfoque es muy común en la industria del software. Lo conocí justamente trabajando como consultor independiente donde de alguna manera estuve obligado interactuar con profesionales de otros ámbitos que "hablan otro idioma". Una vez franqueada la primera barrera de potencial, la interacción es sumamente fructífera ya que no todas las profesiones dan por sentadas las mismas suposiciones y todos terminan enriqueciendo sus capacidades y experiencias.

En general, en términos de emprendedurismo, el *unfair advantage* consiste en que el la herramienta…

- es libre y abierta [83]—remarcando la importancia que esto tiene tanto en la academia como en la industria (sección 4.4.1)
- está pensada como *cloud-first*, concepto que no solamente implica *cloud-friendliness* (sección 4.4.6)
- puede escalar arbitrariamente en paralelo apalancándose en el estándar MPI (sección 4.4.5)
- es un back end diseñado para poder ser manejado con diferentes front ends (figura 4.11)
- sigue la filosofía de diseño Unix (Apéndice C) que es perfectamente aplicable al concepto de *cloud-first*
- provee una interfaz amena a la simulación programática (sección 4.4.3)
- es extremadamente flexible y puede resolver una gran variedad de problemas, desde los más simples con propiedades uniformes hasta los más complejos donde las propiedades de los materiales pueden depender del espacio de maneras no triviales (capítulo 5)
- es a los programas tradicionales (CalculiX, CodeAster) y a las bibliotecas de elementos finitos (Sparselizard, MoFEM) lo que Markdown es a procesadores de texto (Word, Google Docs) y a sistemas de tipografía (TeX), respectivamente (sección B.1)
- está diseñada para que sea posible agregar más tipos de PDEs sin tener que escribir un solver desde cero (sección 4.1.2)

En particular, para las aplicaciones de neutrónica a nivel de núcleo sus características distintivas son que…

1. trabaja sobre mallas no estructuradas
2. puede resolver transporte mediante el método de ordenadas discretas S_N

3. es capaz de resolver problemas de tamaño arbitrario haciendo descomposición de dominio y resolviendo cada parte en un proceso MPI

En el capítulo 1 repasamos las motivaciones para escribir una herramienta computacional que pueda superar las limitaciones de los códigos neutrónicos tradicionales. Con dicho fin, en el capítulo 2 amalgamamos la literatura existente sobre transporte de neutrones para obtener las ecuaciones en derivadas parciales que debemos resolver. Y en el capítulo 3 desarrollamos posibles discretizaciones numéricas para poder resolver efectivamente neutrónica a nivel de núcleo con una (o más) computadoras digitales. Justamente, en el capítulo 4 discutimos y mostramos una de las virtualmente infinitas maneras de diseñar e implementar una herramienta computacional capaz de resolver estas ecuaciones discretizadas. Finalmente, en el capítulo 5 mostramos diez problemas que necesitan al menos una de las características distintivas de FeenoX para poder ser resueltos en forma satisfactoria.

6.1. Trabajos futuros

Es mi deseo que esta tesis dispare un jardín de senderos que se bifurquen para que las ideas y/o las implementaciones discutidas a lo largo de estos cientos de páginas (físicas en la Biblioteca Falicov, lógicas en su versión PDF o web en su versión HTML) no caigan en el ostracismo. En principio, muchas de las tareas pendientes pueden ser encaradas como trabajos académicos y/o proyectos de ingeniería industriales. En algún sentido, el trabajo "futuro" relacionado al gerenciamiento (sea académico o industrial) es más desafiante que los trabajos técnicos listados a continuación ya no sólo que involucran el management de los tres vértices del tradicional triángulo de proyectos

a. costos
b. alcance
c. calidad

sino también, en proyectos nucleares también hay que lidiar con

- recursos humanos especialmente particulares
- gigantescas inercias organizacionales
- impredecibles limitaciones políticas

todos con sus con sus egos y complicaciones, usualmente fruto del hecho de que la industria nuclear extremadamente inestable ya que depende casi exclusivamente de financiamiento y/o incentivos gubernamentales, tanto a nivel local como global. El listado de trabajos futuros sugeridos incluye

- implementación de las ecuaciones de transporte y difusión transitorias, i.e. cinética neutrónica espacial
- comparación cuantitativa entre la aproximación de difusión y el método de ordenadas discretas
  - en trabajos académicos de investigación
  - en modelos de interés industrial
- evaluación de otros conjuntos de cuadraturas no necesariamente de nivel simétrico

- estudios de formas de evitar o mitigar el efecto rayo
- discretización de la coordenada angular con funciones de forma tipo elementos finitos
- incorporación de otras formulaciones neutrónicas
  - P_N
  - SP_N
  - even parity
  - probabilidad de colisiones para cálculo de celda
- instrumentación del código para evaluar su eficiencia y mejorar su performance tanto en CPU como en memoria
  - evaluación de la posibilidad de aplicar el paradigma data-oriented programming
- estudio de factibilidad de utilizar métodos numéricos iterativos para S_N
  - análisis de pre-condicionadores alternativos
  - p-assisted algebraic multi-grid [13]
  - PCPATCH [17]
- mejoramiento de la escalabilidad por paralelización
  - optimización de multi-node MPI
  - análisis de algoritmos de descomposición de dominio para aplicaciones en reactores nucleares de potencia
  - evaluación de la utilización de DMPlex para la distribución de la malla [32]
- desarrollo de interfaces y capas de abstracción
  - plataforma web
  - interfaces gráficas de usuario (GUIs)
  - APIs para lenguajes de scripting (Python, Julia, etc.)
  - "thin clients" para ejecución en la nube
- refinamiento automático de malla[1]
- investigación de otras discretizaciones espaciales
  - elementos de alto orden
  - formulaciones mixtas
  - Galerkin discontinuo
  - volúmenes finitos
  - basada en PetscFE [27]
- aplicaciones a problemas de optimización
  - recocido simulado
  - algoritmos genéticos
  - redes neuronales
  - optimización topológica [89]
- implementación de otras formas numéricas de prescribir condiciones de contorno de Dirichlet multi-punto

[1] Del inglés *Automatic Mesh Refinement*

  - multiplicadores de Lagrange
  - eliminación directa
- otras ecuaciones
  - elasticidad no lineal
  - electromagnetismo
  - mecánica de fluidos
  - termohidráulica 1D
  - CFD
  - aplicaciones a biotecnología
  - acústica
  - etc.
- estudio de implementación de solvers basados en GPUs/APUs
- mejoramiento de la integración continua
  - nuevos tests
  - medición de la cobertura del código
  - análisis sistemático del código con analizadores de memoria tipo valgrind
- estudio de compatibilidad de las secciones eficaces homogeneizadas en celdas estructuradas con su uso en el esquema multi-escala (sección 2.5) con mallas no estructuradas a nivel de núcleo
  - ensambles de elementos combustibles
  - barras de control
  - nubes de boro en inyección de emergencia
- evaluación de la posibilidad de incorporar FeenoX a cadenas de cálculo neutrónico industriales
- acoplamiento con otros códigos de cálculo
  - comunicadores MPI
  - memoria compartida
  - sockets TCP
  - archivo intermedios en almacenamiento tipo RAM-disks
- creación de comunidades libres, abiertas y anti-frágiles
  - académicas
  - industriales
- evaluación de generación de emprendimientos tipo start up susceptibles de ser invertidos y desarrollados en incubadoras como CITES.

Referencias

> Cuando se proclamó que la Biblioteca abarcaba todos los libros, la primera impresión fue de extravagante felicidad. Todos los hombres se sintieron señores de un tesoro intacto y secreto.
> *Jorge Luis Borges, La Biblioteca de Babel, 1941*

[1] Ahrens J, Geveci B, Law C, «ParaView: An End-User Tool for Large Data Visualization», en *Visualization Handbook*, Elesvier, 2005.

[2] Alnæs MS, Logg A, Ølgaard KB, Rognes ME, Wells GN, «Unified Form Language: A Domain-Specific Language for Weak Formulations of Partial Differential Equations», *ACM Trans. Math. Softw.*, vol. 40, n.º 2, mar. 2014, doi: 10.1145/2566630.

[3] ANS, «Argonne Code Center: Benchmark Problem Book», Argonne National Laboratory, ANL-7416 Supplement 2, jun. 1977.

[4] Azmy YY, «The Weighted Diamond-Difference Form of Nodal Transport Methods», *Nuclear Science and Engineering*, vol. 98, n.º 1, pp. 29-40, 1988, doi: 10.13182/NSE88-6.

[5] Babcsány B, «Development of a finite-element-based reactor physics code system for the solution of the simplified P3 approximation to the neutron transport equation», A thesis submitted for the degree of Doctor of Philosophy, Budapest University of Technology; Economics, 2021.

[6] Baker AH, Kolev TzV, Yang UM, «Improving algebraic multigrid interpolation operators for linear elasticity problems», *Numerical Linear Algebra With Applications*, vol. 17, pp. 495-517, 2010, doi: 10.1002/nla.688.

[7] Balay S, Abhyankar S, Adams MF, Benson S, Brown J, Brune P, Buschelman K, Constantinescu E, Dalcin L, Dener A, Eijkhout V, Faibussowitsch J, Gropp WD, Hapla V, Isaac T, Jolivet P, Karpeev D, Kaushik D, Knepley MG, Kong F, Kruger S, May DA, McInnes LC, Mills RT, Mitchell L, Munson T, Roman JE, Rupp K, Sanan P, Sarich J, Smith BF, Zampini S, Zhang H, Zhang H, Zhang J, «PETSc/TAO Users Manual», Argonne National Laboratory, ANL-21/39 - Revision 3.19, 2023. doi: 10.2172/1968587.

[8] Balay S, Gropp WD, McInnes LC, Smith BF, «Efficient Management of Parallelism in Object Oriented Numerical Software Libraries», en *Modern Software Tools in Scientific Computing*, ed. E Arge, AM Bruaset, HP Langtangen, Birkhäuser Press, 1997, pp. 163-202.

[9] Bathe K-J, *Finite Element Procedures*, 2nd ed. Prentice Hall, Pearson Education Inc., 2014.

[10] Beckurts KH, Wirtz K, *Neutron Physics*. Springer-Verlag, 1964.

[11] Brandt A, «Multi-Level Adaptive Solutions to Boundary-Value Problems», *Mathematics of Computation*, vol. 31, n.º 138, p. 333.390, abr. 1977.

[12] Brenner SC, Scott LR, *The Mathematical Theory of Finite Elements Methods*, 3rd ed. Springer, 2008.

[13] Brown J, Barra V, Beams N, Ghaffari L, Knepley M, Moses W, Shakeri R, Stengel K, Thompson JL, Zhang J, «Performance Portable Solid Mechanics via Matrix-Free p-Multigrid». 2022. Disponible en: https://arxiv.org/abs/2204.01722

[14] Clarke L, Glendinning I, Hempel R, «The MPI Message Passing Interface Standard», en *Programming Environments for Massively Parallel Distributed Systems*, ed. KM Decker, RM Rehmann, Basel: Birkhäuser Basel, 1994, pp. 213-218.

[15] Duderstadt JJ, Hamilton LJ, *Nuclear Reactor Analysis*. Wiley, New York, 1976.

[16] Eymard R, Gallouët T, Herbin R, «Handbook of Numerical Analysis», 2000.

[17] Farrell PE, Knepley MG, Mitchell L, Wechsung F, «PCPATCH: Software for the Topological Construction of Multigrid Relaxation Methods», *ACM Trans. Math. Softw.*, vol. 47, n.º 3, jun. 2021, doi: 10.1145/3445791.

[18] Felippa CA, *Introduction to Finite Element Methods*. University of Colorado Boulder, 2004.

[19] Gardner DJ, Reynolds DR, Woodward CS, Balos CJ, «Enabling new flexibility in the SUNDIALS suite of nonlinear and differential/algebraic equation solvers», *ACM Transactions on Mathematical Software (TOMS)*, 2022, doi: 10.1145/3539801.

[20] Geuzaine C, Remacle JF, «Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities», *International Journal for Numerical Methods in Engineering*, vol. 79, n.º 11, pp. 1309-1331, 2009.

[21] Glasstone G S. Bell, *Nuclear Reactor Theory*. Krieger Publishing Company, 1970.

[22] Halton JH, «Algorithm 247: Radical-Inverse Quasi-Random Point Sequence», *Commun. ACM*, vol. 7, n.º 12, pp. 701-702, dic. 1964, doi: 10.1145/355588.365104.

[23] Henry AF, *Nuclear Reactor Analysis*. Cambridge, MIT, 1975.

[24] Hernandez V, Roman JE, Vidal V, «SLEPc: A Scalable and Flexible Toolkit for the Solution of Eigenvalue Problems», *ACM Trans. Math. Software*, vol. 31, n.º 3, pp. 351-362, 2005, doi: 10.1145/1089014.1089019.

[25] Hughes TJR, *The Finite Element Method*. Prentince Hall, 1987.

[26] Jost J, *Partial Differential Equations*, 3rd ed. New York: Springer, 2013.

[27] Kirby RC, «Algorithm 839: FIAT, a New Paradigm for Computing Finite Element Basis Functions», *ACM Transactions on Mathematical Software*, vol. 30, n.º 4, pp. 502-516, 2004, doi: 10.1145/1039813.1039820.

[28] Knuth DE, *The TeXbook*. Addison-Wesley, 1984.

[29] Knuth DE, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, Third. Addison-Wesley, 1997.

[30] Knuth DE, *Selected Papers on Computer Languages*. Center for the Study of Language; Information, 2003.

[31] Lamarsh JR, *Introduction to Nuclear Reactor Theory*. Addisson-Wesley, 1966.

[32] Lange M, Mitchell L, Knepley MG, Gorman GJ, «Efficient mesh management in Firedrake using PETSc-DMPlex», *SIAM Journal on Scientific Computing*, vol. 38, n.º 5, pp. S143-S155, 2016, doi: 10.1137/15M1026092.

[33] Lestani H, «Metodología de análisis neutrónico de celdas de reactores de agua pesada», Tesis de Doctorado en Ingeniería Nuclear, Instituto Balseiro, 2014.

[34] Lewis EE, Miller WF, *Computational Methods of Neutron Transport*. John Wiley; Sons, 1984.

[35] Liu WK, Li S, Park HS, «Eighty Years of the Finite Element Method: Birth, Evolution, and Future», *Archives of Computational Methods in Engineering*, vol. 29, pp. 4431-4453, 2022.

[36] Márquez Damián JI, «Multilevel Acceleration of Neutron Transport Calculations», mathesis, Georgia Institute of Technology, 2007.

[37] Márquez Damián JI, «Desarrollo y aplicaciones de nuevas bibliotecas de secciones eficaces neutrónicas para H_2O, D_2O y HDO», Tesis de Doctorado en Ciencias de la Ingeniería, Instituto Balseiro, 2014.

[38] Martin WR, Yehnert CE, Lorence L, Duderstad JJ, «Phase-Space Finite Element Methods Applied To The First-Order Form Of The Transport Equation», *Annals of Nuclear Energy*, vol. 8, pp. 633-646, 1981.

[39] Nelder JA, Mead R, «A Simplex Method for Function Minimization», *The Computer Journal*, vol. 7, n.º 4, pp. 308-313, ene. 1965, doi: 10.1093/comjnl/7.4.308.

[40] Oberkampf WL, Trucano TG, «Verification and validation benchmarks», *Nuclear Engineering and Design*, vol. 238, pp. 716-743, 2008, Disponible en: https://api.semanticscholar.org/CorpusID:110737903

[41] Park HK, «Coupled Space-Angle Adaptivity And Goal-Oriented Error Control For Radiation Transport Calculations», Tesis doctoral, Georgia Institute of Technology, 2006.

[42] Patel A, Theler J, Levrero-Florencio F, Abboud N, Joshaghani MS, McClennan S, «Scalable cloud-native thermo-mechanical solvers using PETSc», en *PETSc Annual Meeting*, Chicago, 2023.

[43] Pellegrino E, «Desarrollo de un código neutrónico de transporte multigrupo 3D por el método de elementos finitos». Proyecto Integrador de la Carrera de Ingeniería Nuclear, Instituto Balseiro, 2010.

[44] Quarteroni A, *Numerical Models for Differential Problems*. Springer, 2009.

[45] Raouafi H, «Simulation de mécanismes de contrôle de la réactivité inclinés du réacteur SCWR-canadien en utilisant les codes Dragon-5 et Donjon-3», Tesis doctoral, École Polytechnique De Montréal, 2017.

[46] Raymond ES, *The Cathedral and the Bazaar*, 2nd ed. O'Reilly, 2001.

[47] Raymond ES, *The Art of UNIX Programming*. Addison-Wesley, 2003.

[48] Reed WH, «New Difference Schemes for the Neutron Transport Equation», *Nuclear Science and Engineering*, vol. 46, n.º 2, pp. 309-314, 1971, doi: 10.13182/NSE46-309.

[49] Rhodes R, *The making of the atomic bomb*, Twenty-fifth anniversary edition. Simon & Schuster Paperbacks, 2012.

[50] Ritchie D, Thompson K, «The UNIX time-sharing system», *Communications of the ACM*, vol. 7, n.º 17, pp. 365-375, 1974, doi: 10.1145/361011.361061.

[51] Roache PJ, *Verification and Validation in Computational Science and Engineering*. Albuquerque NM: Hermosa Publishers, 1998.

[52] Roman JE, Campos C, Dalcin L, Romero E, Tomas A, «SLEPc Users Manual», D. Sistemes Informàtics i Computació, Universitat Politècnica de València, DSIC-II/24/02 - Revision 3.19, 2023.

[53] Salari K, Knupp P, «Code Verification by the Method of Manufactured Solutions», Sandia Natioal Laboratories, SAMD2000-1444, 2000.

[54] Shepard D, «A two-dimensional interpolation function for irregularly-spaced data», en *Proceedings of the 1968 ACM National Conference*, 1968, pp. 517-524. doi: doi:10.1145/800186.810616.

[55] Sobol IM, «Distribution of points in a cube and approximate evaluation of integrals», *U.S.S.R Comput. Maths. Math. Phys.*, vol. 7, pp. 86-112, 1967.

[56] Sood A, Forster RA, Parsons DK, «Analytical Benchmark Test Set for Criticality Code Verification», Los Alamos National Laboratory, LA-13511, 1999.

[57] Stallman RM, «The GNU Manifesto», *Dr. Dobb's Journal of Software Tools*, vol. 10, n.º 3, 1985.

[58] Stammler RJG, Abbate MJ, *Methods of Steady-State Reactor Physics in Nuclear Design*. Academic Press, 1983.

[59] Steffen M, «A simple method for monotonic interpolation in one dimension», *Astron. Astrophys.*, n.º 239, pp. 443-450, 1990.

[60] [redacted] «Controladores basados en lógica difusa y loops de convección natural caóticos». Proyecto Integrador de la Carrera de Ingeniería Nuclear[redacted], 2007.

[61] [redacted] «Análisis no lineal de inestabilidades en el problema acoplado termohidráulico-neutrónico». Tesis de la Carrera de Maestría en Ingeniería[redacted], 2008.

[62] [redacted] «Difusión de neutrones en mallas no estructuradas: comparación entre volúmenes y elementos finitos». Monografía Final de la materia «Introducción al método de elementos finitos», Universidad de Buenos Aires, 2013.

[63] [redacted] «Unstructured Grids and the Multigroup Neutron Diffusion Equation», *Science and Technology of Nuclear Installations*, vol. 2013:641863, 2013, doi: 10.1155/2013/641863.

[64] [redacted] «Geometric Optimization of Nuclear Reactor Cores», *Mecánica Computacional*, vol. XXXII, n.º Number 32 Optimization and Control, pp. 2659-2709, 2013.

[65] [redacted] «Generalización del código neutrónico PCE para su aplicación a Atucha I», en *Actas de la XLI Reunión Anual de la Asociación Argentina de Tecnología Nuclear*, Buenos Aires, 2014.

[66] [redacted], «On the design basis of a new core-level neutronic code written from scratch», *Mecánica Computacional*, vol. XXXIII, n.º Number 48, Numerical Methods in Reactor Physics (B), pp. 3169-3194, 2014.

[67] [redacted], «Verification by the Method of Manufactured Solutions: application to the steady-state thermal problem», Onscale, 2021.

[68] [redacted], «A cloud-first approach for solving core-level neutron transport», en *8va. Reunión Anual Grupo Argentino de Cálculo y Análisis de Reactores*, 2022.

[69] [redacted], «Verification of PDE solvers with the Method of Manufactured Solutions», en *Ansys TechCon*, Pittsburg, 2023.

[70] [redacted], «Sobre la verificación de un Solver neutrónico con el método de soluciones fabricadas», en *9va. Reunión Anual Grupo Argentino de Cálculo y Análisis de Reactores*, 2023.

[71] [redacted], «On the stability of the point reactor kinetics equations», *Nuclear Engineering and Design*, vol. 240, n.º 6, pp. 1443-1449, jun. 2010, doi: 10.1016/j.nucengdes.2010.03.007.

[72] [redacted], «Optimización de parametros en reactores de potencia: base de diseño del codigo neutrónico milonga», *Reunion Anual de la Asociacion Argentina de Tecnologia Nuclear*, vol. XXXVII, 2010.

[73] [redacted] «Solution of the 2D IAEA PWR Benchmark with the neutronic code milonga», *Actas de la Reunión Anual de la Asociación Argentina de Tecnología Nuclear*, vol. XXXVIII, 2011.

[74] [redacted] Gómez Omil JP, Mazzantini O, «A coupled scheme for the deterministic safety transient analysis of the Atucha I Nuclear Power Plant», *Mecánica Computacional*, vol. Volume XXXIII, n.º Number 45 Computational Aplications in Nuclear Technology (B), pp. 2939-2955, 2014.

[75] [redacted] Gómez Omil JP, Pellegrino E, «A shared-memory-based coupling scheme for modeling the behavior of a Nuclear Power Plant core», *Mecánica Computacional*, vol. XXXII, n.º Number 18 Multiphysics, pp. 1501-1517, 2013.

[76] [redacted] Mazzantini O, «Desarrollo de capacidades locales para ingeniería de licenciamiento de centrales nucleares: evaluación de los efectos de la inyección de boro de emergencia», en *Congreso Ingeniería 2014, Buenos Aires*, Buenos Aires, 2014.

[77] [redacted] Mazzantini O, Schivo M, Cesare JD, Garbero R, Rivero M, «A coupled calculation suite for Atucha II operational transients analysis», *Science and Technology of Nuclear Installations*, vol. 2011:785304, 2011, doi: 10.1155/2011/785304.

[78] [redacted] Roqueta D, Tarazaga A, «Cálculo de Secciones Eficaces Condensadas a 2 Grupos, para Simulación de Transitorios con Inyección Rápida de Boro Enriquecido», en *Actas de la XLI Reunión Anual de la Asociación Argentina de Tecnología Nuclear*, Buenos Aires, 2014.

[79] [redacted] Schivo M, «Estimación del coeficiente de reactividad por temperatura del combustible en reactores tipo Atucha mediante ajuste de modelos matematicos a mediciones experimentales», en *Actas de la XL Reunión Anual de la Asociación Argentina de Tecnología Nuclear*, Buenos Aires, 2013.

[80] [redacted], Schivo M, Salom G, «Estimación del coeficiente de reactividad por temperatura del combustible de la Central Nuclear Atucha II a partir de mediciones de flujo neutrónico», en *Actas de la XLI Reunión Anual de la Asociación Argentina de Tecnología Nuclear*, Buenos Aires, 2014.

[81] [redacted], Tarazaga A, Roqueta D, «Desarrollo de la neutrónica del segundo sistema de extinción de la Central Nuclear Atucha I para su implementación en un esquema de acople completo de control, termohidráulica y neutrónica», en *Actas de la XLI Reunión Anual de la Asociación Argentina de Tecnología Nuclear*, Buenos Aires, 2014.

[82] [redacted], Vasconcelos V, Santos A, Campolina D, Pereira C, «Coupled unstructured fine-mesh neutronics and thermal-hydraulics methodology using open software: A proof-of-concept», *Annals of Nuclear Energy*, vol. 115, pp. 173-185, 2018, doi: 10.1016/j.anucene.2018.01.021.

[83] [redacted] «FeenoX: a cloud-first finite-element(ish) computational engineering tool», *Journal of Open Source Software*, vol. 9, n.º 95, p. 5846, mar. 2024, doi: 10.21105/joss.05846.

[84] Trobec R, Slivnik B, Bulić P, Robič B, *Introduction to Parallel Computing*. Switzerland: Springer, 2018.

[85] Vadén T, Stallman RM, *The Hacker Community and Ethics: An Interview with Richard M. Stallman*. Tampere University Press, 2002.

[86] Van Criekingen S, «Mixed-Hybrid Discretization Methods for the Linear Transport Equation», Tesis doctoral, Northwestern University, 2004.

[87] Vasconcelos Araújo Silva V, «Acoplamento neutrônico e termo-hidráulico usando os códigos milonga e OpenFOAM: uma abordagem com software livre», Pós-Graduação Em Ciências E Técnicas Nucleares Doutor em Ciências e Técnicas Nucleares, Universidade Federal De Minas Gerais, 2016.

[88] Williams S, *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly, 2002.

[89] Winter AEP, «Optimización térmica del reflector de un reactor nuclear», mathesis, Instituto Balseiro, 2023.

[90] Worlton WJ, Voorhees EA, «Recent Developments in Computers and Their Implication for Reactor Calculations», en *Proceedings of the Conference on the Applications to Reactor Problems*, American Nuclear Society, 1965.

[91] Zienkiewicz OC, Taylor RL, Zhu JZ, *The Finite Element Method: its basis and fundamentals*, 6th ed., vol. 1. Elsevier, 2005.

The Journal of Open Source Software

FeenoX: a cloud-first finite-element(ish) computational engineering tool

[redacted] [1,2]

1 [redacted] Argentina 2 [redacted], Argentina

DOI: 10.21105/joss.05846

Software

- Review
- Repository
- Archive

Editor: Kevin M. Moerman

Reviewers:

- @vijaysm
- @AnjaliSandip
- @chennachaos

Submitted: 27 July 2023
Published: 16 March 2024

Summary

FeenoX is a cloud-first free no-X uniX-like finite-element(ish) computational engineering tool designed to solve engineering-related problems using cloud servers in parallel in such a way that the problem is defined in a plain-text near-English self descriptive input file read at run time, without requiring further user intervention after the invocation. FeenoX meets fictitious-yet-plausible Software Requirement Specifications (SRS). The FeenoX Software Design Specifications address each requirement of the SRS. FeenoX provides a set of common extents, capabilities and usefulness but offers different features (following slightly different spirits) for industry engineers, Unix hackers and academic researchers. The main features of this design basis are

- The tool has to be an already-compiled program (not a library) so regular users do not have to compile anything to solve a problem.
- Simple problems ought to need simple input files.
- There should be a one-to-one correspondence between the problem definition and FeenoX's input file, as illustrated in fig. 1.
- There should be an extension mechanism to allow hackers and researchers to add new partial differential equations to the tool.

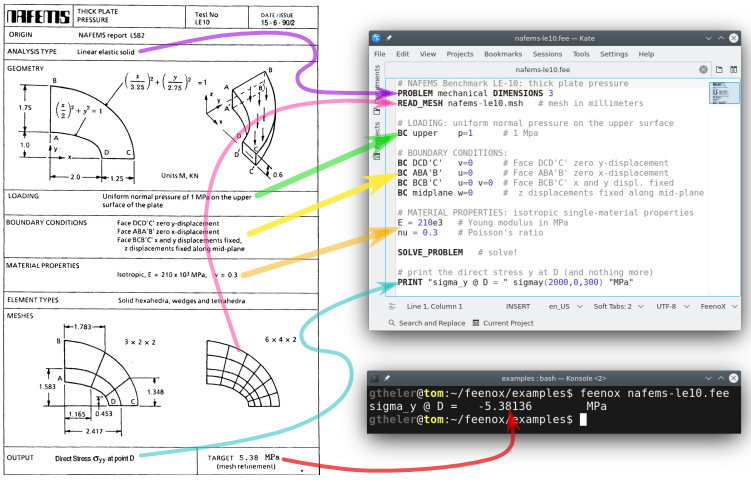


Figure 1: The NAFEMS LE10 problem statement (Finite Element Methods & Standards (Great Britain), 1990) and the corresponding FeenoX input illustrating the one-to-one correspondence between the two.

[redacted] (2024). FeenoX: a cloud-first finite-element(ish) computational engineering tool. *Journal of Open Source Software*, *9*(95), 5846. https://doi.org/10.21105/joss.05846.

The Journal of Open Source Software

Statement of need

Open-source finite-element tools are either

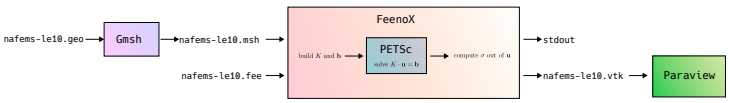
a. libraries which need code to use them such as
   - Sparselizard (Halbach, 2017)
   - MoFEM (Kaczmarczyk et al., 2020)
   - FEniCS (Baratta et al., 2023)
   - MFEM (Anderson et al., 2021)
b. end-user programs which need a GUI such as
   - CalculiX
   - CodeAster

FeenoX sits in the middle. First, it can solve

- Basic mathematics
- Systems of ODEs/DAEs
- Laplace's equation
- Heat conduction
- Linear elasticity
- Modal analysis
- Neutron diffusion
- Neutron discrete ordinates

Second, it is the only free and open-source tool that satisfies the Software Requirement Specifications, including that…

- in order to solve a problem one needs to prepare a (relatively) simple input file (not a script nor a deck) which is read at run-time (not code which calls a library). For example, considering the NAFEMS LE10 Benchmark problem from fig. 1, FeenoX works as two "glue layers" (Raymond, 2003)
  1. between the mesher Gmsh (Geuzaine & Remacle, 2009) and the PETSc library (Balay et al., 1997, 2023)
  2. between the PETSc library and a post-processor such as Paraview (Ayachit, 2015)



- these input files can expand generic command-line options using Bash syntax as $1, $2, etc., which allow parametric or optimization loops driven by higher-level scripts.
- for solving partial differential equations (PDEs), the input file has to refer to at least one Gmsh `.msh` file that defines the domain where the PDE is solved.
- the material properties and boundary conditions are defined using physical groups and not individual nodes nor elements, so the input file is independent of the mesh and thus can be tracked with Git to increase traceability and repeatability.
- it follows the Unix philosophy (Raymond, 2003) which, among others, separates policy from mechanism rendering FeenoX as a natural choice for web-based interfaces like CAEplex (fig. 2).

[redacted] (2024). FeenoX: a cloud-first finite-element(ish) computational engineering tool. *Journal of Open Source Software*, *9*(95), 5846. 2 https://doi.org/10.21105/joss.05846.

The Journal of Open Source Software

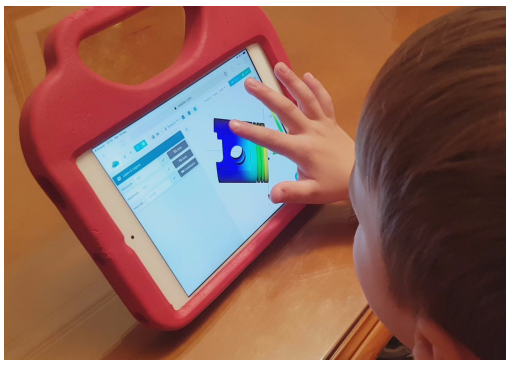


Figure 2: CAEplex is a web-based interface to solve thermo-mechanical problems in the cloud that uses FeenoX as the back end.

FeenoX tries to achieve its goals by…

- standing on both ethical (since it is free) and technical (since it is open source) grounds while interacting with other free and open operating systems, libraries, compilers and pre and post-processing tools, thus encouraging science and engineering to shift from privative environments into the free world.
- leveraging the Unix programming philosophy to come up with a cloud-first tool suitable to be automatically deployed and serve as the back end of web-based interfaces such as CAEplex.
- providing a ready-to-run program that reads an input file at run time (and not a library that has to be linked for each particular problem to be solved) as a deliberate design decision discussed in the Software Design Specifications.
- designing and implementing an extensibility mechanism to allow hackers and/or academics to add new PDE formulations by adding a new subdirectory to `src/pdes` in the repository and then
  a. re-bootstrapping with `autogen.sh`,
  b. re-configuring with `configure`, and
  c. re-compiling with `make`

In effect, FeenoX provides a general mathematical framework to solve PDEs with a bunch of entry points (as C functions) where new types of PDEs (e.g. electromagnetism, fluid mechanics, etc.) can be added to the set of what FeenoX can solve. This general framework provides means to

- parse the input file, handle command-line arguments, read mesh files, assign variables, evaluate conditionals, write results, etc.

```
PROBLEM laplace 2D
READ_MESH square-$1.msh
[...]
WRITE_RESULTS FORMAT vtk
```

- handle material properties given as algebraic expressions involving pointwise-defined functions of space, temperature, time, etc.

```
MATERIAL steel     E=210e3*(1-1e-3*(T(x,y,z)-20))   nu=0.3
MATERIAL aluminum  E=69e3                           nu=7/25
```

(2024). FeenoX: a cloud-first finite-element(ish) computational engineering tool. *Journal of Open Source Software*, *9*(95), 5846. https://doi.org/10.21105/joss.05846.

- read problem-specific boundary conditions as algebraic expressions

```
sigma = 5.670374419e-8  # W m^2 / K^4 as in wikipedia
e = 0.98      # non-dimensional
T0 = 1000     # K
Tinf = 300    # K

BC left  T=T0
BC right q=sigma*e*(Tinf^4-T(x,y,z)^4)
```

- access shape functions and its derivatives evaluated either at Gauss points or at arbitrary locations for computing elementary contributions to
  - stiffness matrix
  - mass matrix
  - right-hand side vector

  For example, this snippet would build the elemental stiffness matrix for the Laplace problem:

```c
int build_laplace_Ki(element_t *e, unsigned int q) {
  double wdet = feenox_fem_compute_w_det_at_gauss(e, q);
  gsl_matrix *B = feenox_fem_compute_B_at_gauss(e, q);
  feenox_call(feenox_blas_BtB_accum(B, wdet, feenox.fem.Ki));
  return FEENOX_OK;
}
```

  The calls for computing the weights and the matrices with the shape functions and/or their derivatives currently support first and second-order iso-geometric elements, but other element types can be added as well. More complex cases involving non-uniform material properties, volumetric sources, etc. can be found in the examples, tutorials and tests.

- solve the discretized equations using the appropriate PETSc (Balay et al., 1997, 2023) or SLEPc (Hernandez et al., 2005; Roman et al., 2023) objects, i.e.
  - KSP for linear static problems
  - SNES for non-linear static problems
  - TS for transient problems
  - EPS for eigenvalue problems

The particular functions that implement each problem type are located in subdirectories `src/pdes`, namely

- `laplace`
- `thermal`
- `mechanical`
- `modal`
- `neutron_diffusion`
- `neutron_sn`

Researchers with both knowledge of mathematical theory of finite elements and programming skills might, with the aid of the community, add support for other PDEs. They might do that by using one of these directories (say `laplace`) as a template and

1. replace every occurrence of `laplace` in symbol names with the name of the new PDE
2. modify the initialization functions in `init.c` and set
   - the names of the unknowns
   - the names of the material properties
   - the mathematical type and characteristics of problem
   - etc.

(2024). FeenoX: a cloud-first finite-element(ish) computational engineering tool. *Journal of Open Source Software*, *9*(95), 5846. https://doi.org/10.21105/joss.05846.

The Journal of Open Source Software

3. modify the contents of the elemental matrices in `bulk.c` in the FEM formulation of the problem being added
4. modify the contents of how the boundary conditions are parsed and set in `bc.c`
5. re-run `autogen.sh`, `./configure` and `make` to get a FeenoX executable with support for the new PDE.

The addition of non-trivial PDEs is not straightforward, but possible. The programming guide contains further details about how to contribute to the code base.

Conclusions

FeenoX's main goal is to keep things simple as possible from the user's point of view without sacrificing flexibility. There exist other tools which are similar in functionality but differ in the way the problem is set up. For example, FeniCSx uses the Unified Form Language where the PDE being solved has to be written by the user in weak form (Alnæs et al., 2014). This approach is very flexible, but even simple problems end up with non-trivial input files so it does not fulfill the first requirement stated in the summary. As simple as it is, FeenoX is still pretty flexible. A proof of this fact is that its applications range from coupling neutronics with CFD in nuclear reactors (Vasconcelos et al., 2018) to providing a back end to web-based thermo-mechanical solvers.

References

Alnæs, M. S., Logg, A., Ølgaard, K. B., Rognes, M. E., & Wells, G. N. (2014). Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Trans. Math. Softw.*, *40*(2). https://doi.org/10.1145/2566630

Anderson, R., Andrej, J., Barker, A., Bramwell, J., Camier, J.-S., Cerveny, J., Dobrev, V., Dudouit, Y., Fisher, A., Kolev, Tz., Pazner, W., Stowell, M., Tomov, V., Akkerman, I., Dahm, J., Medina, D., & Zampini, S. (2021). MFEM: A modular finite element methods library. *Computers & Mathematics with Applications*, *81*, 42—74. https://doi.org/10.1016/j.camwa.2020.06.009

Ayachit, U. (2015). *The ParaView guide: A parallel visualization application*. Kitware. ISBN: 9781930934306

Balay, S., Abhyankar, S., Adams, M. F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E., Dalcin, L., Dener, A., Eijkhout, V., Faibussowitsch, J., Gropp, W. D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, D., Kaushik, D., Knepley, M. G., … Zhang, J. (2023). *PETSc/TAO users manual* (ANL-21/39 - Revision 3.19). Argonne National Laboratory. https://doi.org/10.2172/1968587

Balay, S., Gropp, W. D., McInnes, L. C., & Smith, B. F. (1997). Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, & H. P. Langtangen (Eds.), *Modern software tools in scientific computing* (pp. 163–202). Birkhäuser Press. https://doi.org/10.1007/978-1-4612-1986-6_8

Baratta, I. A., Dean, J. P., Dokken, J. S., Habera, M., Hale, J. S., Richardson, C. N., Rognes, M. E., Scroggs, M. W., Sime, N., & Wells, G. N. (2023). *DOLFINx: The next generation FEniCS problem solving environment*. preprint. https://doi.org/10.5281/zenodo.10447666

Finite Element Methods & Standards (Great Britain), N. A. for. (1990). *NAFEMS: The standard NAFEMS benchmarks*. NAFEMS. https://books.google.com.ar/books?id=--qwHAAACAAJ

Geuzaine, C., & Remacle, J.-F. (2009). Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical*

Methods in Engineering, *79*(11), 1309—1331. https://doi.org/10.1016/j.camwa.2020.06.009

Halbach, A. (2017). *Sparselizard—the user friendly finite element C++ library* [PhD thesis]. Université de Liège—Dép. d'électric., électron. et informat. (Inst.Montefiore).

Hernandez, V., Roman, J. E., & Vidal, V. (2005). SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, *31*(3), 351–362. https://doi.org/10.1145/1089014.1089019

Kaczmarczyk, Ł., Ullah, Z., Lewandowski, K., Meng, X., Zhou, X.-Y., Athanasiadis, I., Nguyen, H., Chalons-Mouriesse, C.-A., Richardson, E., Miur, E., Shvarts, A., Wakeni, M., & Pearce, C. (2020). MoFEM: An open source, parallel finite element library. *The Journal of Open Source Software*. https://doi.org/10.21105/joss.01441

Raymond, E. S. (2003). *The art of UNIX programming*. Addison-Wesley.

Roman, J. E., Campos, C., Dalcin, L., Romero, E., & Tomas, A. (2023). *SLEPc users manual* (DSIC-II/24/02 - Revision 3.19). D. Sistemes Informàtics i Computació, Universitat Politècnica de València.

Vasconcelos, V., Santos, A., Campolina, D., [redacted] & Pereira, C. (2018). Coupled unstructured fine-mesh neutronics and thermal-hydraulics methodology using open software: A proof-of-concept. *Annals of Nuclear Energy*, *115*, 173–185. https://doi.org/10.1016/j.anucene.2018.01.021

A. Software Requirements Specification for an Engineering Computational Tool

An imaginary (a thought experiment if you will) "Request for Quotation" issued by a fictitious agency asking for vendors to offer a free and open source cloud-based computational tool to solve engineering problems. This (imaginary but plausible) Software Requirements Specification document describes the mandatory features this tool ought to have and lists some features which would be nice the tool had, following current state-of-the-art methods and technologies.

A.1. Introduction

A computational tool (herein after referred to as *the tool*) specifically designed to be executed in arbitrarily-scalable remote servers (i.e. in the cloud) is required in order to solve engineering problems following the current state-of-the-art methods and technologies impacting the high-performance computing world. This (imaginary but plausible) Software Requirements Specification document describes the mandatory features this tool ought to have and lists some features which would be nice the tool had. Also it contains requirements and guidelines about architecture, execution and interfaces in order to fulfill the needs of cognizant engineers as of the 2020s.

On the one hand, the tool should allow to solve industrial problems under stringent efficiency (section A.2.3) and quality (section A.4) requirements. It is therefore mandatory to be able to assess the source code for

- independent verification, and/or
- performance profiling, and/or
- quality control

by qualified third parties from all around the world. Hence, it has to be *open source* according to the definition of the Open Source Initiative.

On the other hand, the initial version of the tool is expected to provide a basic functionality which might be extended (section A.1.1 and section A.2.6) by academic researchers and/or professional programmers. It thus should also be *free*—in the sense of freedom, not in the sense of price—as defined by the Free Software Foundation. There is no requirement on the pricing scheme, which is up to the vendor to define in the offer along with the detailed licensing terms. These should allow users to solve their problems the way they need and, eventually, to modify and improve the tool to suit their needs. If they cannot program themselves, they should have the *freedom* to hire somebody to do it for them.

A.1.1. Objective

The main objective of the tool is to be able to solve engineering problems which are usually casted as differential-algebraic equations (DAEs) or partial differential equations (PDEs), such as

- heat conduction
- mechanical elasticity
- structural modal analysis
- mechanical frequency studies
- electromagnetism
- chemical diffusion
- process control dynamics
- computational fluid dynamics
- …

on one or more mainstream cloud servers, i.e. computers with hardware and operating systems (further discussed in section A.2) that allows them to be available online and accessed remotely either interactively or automatically by other computers as well. Other architectures such as high-end desktop personal computers or even low-end laptops might be supported but they should not the main target (i.e. the tool has to be cloud-first but laptop-friendly).

The initial version of the tool must be able to handle a subset of the above list of problem types. Afterward, the set of supported problem types, models, equations and features of the tool should grow to include other models as well, as required in section A.2.6.

A.1.2. Scope

The tool should allow users to define the problem to be solved programmatically. That is to say, the problem should be completely defined using one or more files either…

a. specifically formatted for the tool to read such as JSON or a particular input format (historically called input decks in punched-card days), and/or
b. written in an high-level interpreted language such as Python or Julia.

Once the problem has been defined and passed on to the solver, no further human intervention should be required.

It should be noted that a graphical user interface is *not* required. The tool may include one, but it should be able to run without needing any interactive user intervention rather than the preparation of a set of input files. Nevertheless, the tool might *allow* a GUI to be used. For example, for a basic usage involving simple cases, a user interface engine should be able to create these problem-definition files in order to give access to less advanced users to the tool using a desktop, mobile and/or web-based interface in order to run the actual tool without needing to manually prepare the actual input files.

However, for general usage, users should be able to completely define the problem (or set of problems, i.e. a parametric study) they want to solve in one or more input files and to obtain one or more output files containing the desired results, either a set of scalar outputs (such as maximum stresses or mean

temperatures), and/or a detailed time and/or spatial distribution. If needed, a discretization of the domain may to be taken as a known input, i.e. the tool is not required to create the mesh as long as a suitable mesher can be employed using a similar workflow as the one specified in this SRS.

The tool should define and document (section A.4.6) the way the input files for a solving particular problem are to be prepared (section A.3.1) and how the results are to be written (section A.3.2). Any GUI, pre-processor, post-processor or other related graphical tool used to provide a graphical interface for the user should integrate in the workflow described in the preceding paragraph: a pre-processor should create the input files needed for the tool and a post-processor should read the output files created by the tool.

A.2. Architecture

The tool must be aimed at being executed unattended on remote servers which are expected to have a mainstream (as of the 2020s) architecture regarding operating system (GNU/Linux variants and other Unix-like OSes) and hardware stack, such as

- a few Intel-compatible or ARM-like CPUs per host
- a few levels of memory caches
- a few gigabytes of random-access memory
- several gigabytes of solid-state storage

It should successfully run on

- bare-metal
- virtual servers
- containerized images

using standard compilers, dependencies and libraries already available in the repositories of most current operating systems distributions.

Preference should be given to open source compilers, dependencies and libraries. Small problems might be executed in a single host but large problems ought to be split through several server instances depending on the processing and memory requirements. The computational implementation should adhere to open and well-established parallelization standards.

Ability to run on local desktop personal computers and/laptops is not required but suggested as a mean of giving the opportunity to users to test and debug small coarse computational models before launching the large computation on a HPC cluster or on a set of scalable cloud instances. Support for non-GNU/Linux operating systems is not required but also suggested.

Mobile platforms such as tablets and phones are not suitable to run engineering simulations due to their lack of proper electronic cooling mechanisms. They are suggested to be used to control one (or more) instances of the tool running on the cloud, and even to pre and post process results through mobile and/or web interfaces.

A.2.1. Deployment

The tool should be easily deployed to production servers. Both

a. an automated method for compiling the sources from scratch aiming at obtaining optimized binaries for a particular host architecture should be provided using a well-established procedures, and
b. one (or more) generic binary version aiming at common server architectures should be provided.

Either option should be available to be downloaded from suitable online sources, either by real people and/or automated deployment scripts.

A.2.2. Execution

It is mandatory to be able to execute the tool remotely, either with a direct action from the user or from a high-level workflow which could be triggered by a human or by an automated script. Since it is required for the tool to be able to be run distributed among different servers, proper means to perform this kind of remote executions should be provided. The calling party should be able to monitor the status during run time and get the returned error level after finishing the execution.

The tool shall provide means to perform parametric computations by varying one or more problem parameters in a certain prescribed way such that it can be used as an inner solver for an outer-loop optimization tool. In this regard, it is desirable that the tool could compute scalar values such that the figure of merit being optimized (maximum temperature, total weight, total heat flux, minimum natural frequency, maximum displacement, maximum von Mises stress, etc.) is already available without needing further post-processing.

A.2.3. Efficiency

As required in the previous section, it is mandatory to be able to execute the tool on one or more remote servers. The computational resources needed from this server, i.e. costs measured in

- CPU/GPU time
- random-access memory
- long-term storage
- etc.

needed to solve a problem should be comparable to other similar state-of-the-art cloud-based script-friendly finite-element tools.

A.2.4. Scalability

The tool ought to be able to start solving small problems first to check the inputs and outputs behave as expected and then allow increasing the problem size up in order to achieve to the desired accuracy of the results. As mentioned in section A.2, large problem should be split among different computers to be able to solve them using a finite amount of per-host computational power (RAM and CPU).

A.2.5. Flexibility

The tool should be able to handle engineering problems involving different materials with potential spatial and time-dependent properties, such as temperature-dependent thermal expansion coefficients and/or non-constant densities. Boundary conditions must be allowed to depend on both space and time as well, like non-uniform pressure loads and/or transient heat fluxes.

A.2.6. Extensibility

It should be possible to add other problem types casted as PDEs (such as the Schröedinger equation) to the tool using a reasonable amount of time by one or more skilled programmers. The tool should also allow new models (such as non-linear stress-strain constitutive relationships) to be added as well.

A.2.7. Interoperability

A mean of exchanging data with other computational tools complying to requirements similar to the ones outlined in this document. This includes pre and post-processors but also other computational programs so that coupled calculations can be eventually performed by efficiently exchanging information between calculation codes.

A.3. Interfaces

The tool should be able to allow remote execution without any user intervention after the tool is launched. To achieve this goal it is required that the problem should be completely defined in one or more input files and the output should be complete and useful after the tool finishes its execution, as already required. The tool should be able to report the status of the execution (i.e. progress, errors, etc.) and to make this information available to the user or process that launched the execution, possibly from a remote location.

A.3.1. Problem input

The problem should be completely defined by one or more input files. These input files might be

a. particularly formatted files to be read by the tool in an *ad-hoc* way, and/or
b. source files for interpreted languages which can call the tool through and API or equivalent method, and/or
c. any other method that can fulfill the requirements described so far.

Preferably, these input files should be plain ASCII files in order to allow to manage changes using distributed version control systems such as Git. If the tool provides an API for an interpreted language such as Python, then the Python source used to solve a particular problem should be Git-friendly. It is recommended not to track revisions of mesh data files but of the source input files, i.e. to track the mesher's input and not the mesher's output. Therefore, it is recommended not to mix the problem definition with the problem mesh data.

It is not mandatory to include a GUI in the main distribution, but the input/output scheme should be such that graphical pre and post-processing tools can create the input files and read the output files so as to allow third parties to develop interfaces. It is recommended to design the workflow as to make it possible for the interfaces to be accessible from mobile devices and web browsers.

It is expected that 80% of the problems need 20% of the functionality. It is acceptable if only basic usage can be achieved through the usage of graphical interfaces to ease basic usage at first. Complex problems involving non-trivial material properties and boundary conditions not be treated by a GUI and only available by needing access to the input files.

A.3.2. Results output

The output ought to contain useful results and should not be cluttered up with non-mandatory information such as ASCII art, notices, explanations or copyright notices. Since the time of cognizant engineers is far more expensive than CPU time, output should be easily interpreted by either a human or, even better, by other programs or interfaces—especially those based in mobile and/or web platforms. Open-source formats and standards should be preferred over privative and ad-hoc formatting to encourage the possibility of using different workflows and/or interfaces.

A.4. Quality assurance

Since the results obtained with the tool might be used in verifying existing equipment or in designing new mechanical parts in sensitive industries, a certain level of software quality assurance is needed. Not only are best-practices for developing generic software such as

- employment of a version control system,
- automated testing suites,
- user-reported bug tracking support.
- etc.

required, but also since the tool falls in the category of engineering computational software, verification and validation procedures are also mandatory, as discussed below. Design should be such that governance of engineering data including problem definition, results and documentation can be efficiently performed using state-of-the-art methodologies, such as distributed control version systems

A.4.1. Reproducibility and traceability

The full source code and the documentation of the tool ought to be maintained under a control version system. Whether access to the repository is public or not is up to the vendor, as long as the copying conditions are compatible with the definitions of both free and open source software from the FSF and the OSI, respectively as required in section A.1.

In order to be able to track results obtained with different version of the tools, there should be a clear release procedure. There should be periodical releases of stable versions that are required

- not to raise any warnings when compiled using modern versions of common compilers (e.g. GNU, Clang, Intel, etc.)
- not to raise any errors when assessed with dynamic memory analysis tools (e.g. Valgrind) for a wide variety of test cases
- to pass all the automated test suites as specified in section A.4.2

These stable releases should follow a common versioning scheme, and either the tarballs with the sources and/or the version control system commits should be digitally signed by a cognizant responsible. Other unstable versions with partial and/or limited features might be released either in the form of tarballs or made available in a code repository. The requirement is that unstable tarballs and main (a.k.a. trunk) branches on the repositories have to be compilable. Any feature that does not work as expected or that does not even compile has to be committed into develop branches before being merge into trunk.

If the tool has an executable binary, it should be able to report which version of the code the executable corresponds to. If there is a library callable through an API, there should be a call which returns the version of the code the library corresponds to.

It is recommended not to mix mesh data like nodes and element definition with problem data like material properties and boundary conditions so as to ease governance and tracking of computational models and the results associated with them. All the information needed to solve a particular problem (i.e. meshes, boundary conditions, spatially-distributed material properties, etc.) should be generated from a very simple set of files which ought to be susceptible of being tracked with current state-of-the-art version control systems. In order to comply with this suggestion, ASCII formats should be favored when possible.

A.4.2. Automated testing

A mean to automatically test the code works as expected is mandatory. A set of problems with known solutions should be solved with the tool after each modification of the code to make sure

these changes still give the right answers for the right questions and no regressions are introduced. Unit software testing practices like continuous integration and test coverage are recommended but not mandatory.

The tests contained in the test suite should be

- varied,
- diverse, and
- independent

Due to efficiency issues, there can be different sets of tests (e.g. unit and integration tests, quick and thorough tests, etc.) Development versions stored in non-main branches can have temporarily-failing tests, but stable versions have to pass all the test suites.

A.4.3. Bug reporting and tracking

A system to allow developers and users to report bugs and errors and to suggest improvements should be provided. If applicable, bug reports should be tracked, addressed and documented. User-provided suggestions might go into the back log or TO-DO list if appropriate.

Here, "bug and errors" mean failure to

- compile on supported architectures,
- run (unexpected run-time errors, segmentation faults, etc.)
- return a correct result

A.4.4. Verification

Verification, defined as

> The process of determining that a model implementation accurately represents the developer's conceptual description of the model and the solution to the model.

i.e. checking if the tool is solving right the equations, should be performed before applying the code to solve any industrial problem. Depending on the nature and regulation of the industry, the verification guidelines and requirements may vary. Since it is expected that code verification tasks could be performed by arbitrary individuals or organizations not necessarily affiliated with the tool vendor, the source code should be available to independent third parties. In this regard, changes in the source code should be controllable, traceable and well documented.

Even though the verification requirements may vary among problem types, industries and particular applications, a common method to verify the code is to compare solutions obtained with the tool with known exact solutions or benchmarks. It is thus mandatory to be able to compare the results with analytical solutions, either internally in the tool or through external libraries or tools. This approach is called the Method of Exact Solutions and it is the most widespread scheme for verifying computational software, although it does not provide a comprehensive method to verify the whole spectrum of features. In any case, the tool's output should be susceptible of being post-processed and

analyzed in such a way to be able to determine the order of convergence of the numerical solution as compared to the exact one.

Another possibility is to follow the Method of Manufactured Solutions, which does address all the shortcomings of MES. It is highly encouraged that the tool allows the application of MMS for software verification. Indeed, this method needs a full explanation of the equations solved by the tool, up to the point that a report from Sandia National Labs says that

> Difficulties in determination of the governing equations arises frequently with commercial software, where some information is regarded as proprietary. If the governing equations cannot be determined, we would question the validity of using the code.

To enforce the availability of the governing equations, the tool has to be open source as required in section A.1 and well documented as required in section A.4.6.

A report following either the MES and/or MMS procedures has to be prepared for each type of equation that the tool can solve. The report should show how the numerical results converge to the exact or manufactured results with respect to the mesh size or number of degrees of freedom. This rate should then be compared to the theoretical expected order.

Whenever a verification task is performed and documented, at least one of the cases should be added to the test suite. Even though the verification report must contain a parametric mesh study, a single-mesh case is enough to be added to the test suite. The objective of the tests defined in section A.4.2 is to be able to detect regressions which might have been inadvertently introduced in the code and not to do any actual verification. Therefore a single-mesh case is enough for the test suites.

A.4.5. Validation

As with verification, for each industrial application of the tool there should be a documented procedure to perform a set of validation tests, defined as

> The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model.

i.e. checking that the right equations are being solved by the tool. This procedure should be based on existing industry standards regarding verification and validation such as ASME, AIAA, IAEA, etc. There should be a procedure for each type of physical problem (thermal, mechanical, thermo-mechanical, nuclear, etc.) and for each problem type when a new

- geometry,
- mesh type,
- material model,
- boundary condition,
- data interpolation scheme

or any other particular application-dependent feature is needed.

A report following the validation procedure defined above should be prepared and signed by a responsible engineer in a case-by-case basis for each particular field of application of the tool. Verification can be performed against

- known analytical results, and/or
- other already-validated tools following the same standards, and/or
- experimental results.

A.4.6. Documentation

Documentation should be complete and cover both the user and the developer point of view. It should include a user manual adequate for both reference and tutorial purposes. Other forms of simplified documentation such as quick reference cards or video tutorials are not mandatory but highly recommended. Since the tool should be extendable (section A.2.6), there should be a separate development manual covering the programming design and implementation, explaining how to extend the code and how to add new features. Also, as non-trivial mathematics which should be verified are expected, a thorough explanation of what equations are taken into account and how they are solved is required.

It should be possible to make the full documentation available online in a way that it can be both printed in hard copy and accessed easily from a mobile device. Users modifying the tool to suit their own needs should be able to modify the associated documentation as well, so a clear notice about the licensing terms of the documentation itself (which might be different from the licensing terms of the source code itself) is mandatory. Tracking changes in the documentation should be similar to tracking changes in the code base. Each individual document ought to explicitly state to which version of the tool applies. Plain ASCII formats should be preferred. It is forbidden to submit documentation in a non-free format.

The documentation shall also include procedures for

- reporting errors and bugs
- releasing stable versions
- performing verification and validation studies
- contributing to the code base, including
  - code of conduct
  - coding styles
  - variable and function naming conventions

B. FeenoX Software Design Specification

This Software Design Specification (SDS) document applies to an imaginary Software Requirements Specification (SRS) document issued by a fictitious agency asking for vendors to offer a free and open source cloud-based computational tool to solve engineering problems. The latter can be seen as a "Request for Quotation" and the former as an offer for the fictitious tender. Each section of this SDS addresses one section of the SRS. The original text from the SRS is shown quoted at the very beginning before the actual SDS content.

B.1. Introduction

> A computational tool (herein after referred to as *the tool*) specifically designed to be executed in arbitrarily-scalable remote servers (i.e. in the cloud) is required in order to solve engineering problems following the current state-of-the-art methods and technologies impacting the high-performance computing world. This (imaginary but plausible) Software Requirements Specification document describes the mandatory features this tool ought to have and lists some features which would be nice the tool had. Also it contains requirements and guidelines about architecture, execution and interfaces in order to fulfill the needs of cognizant engineers as of the 2020s.
>
> On the one hand, the tool should allow to solve industrial problems under stringent efficiency (section A.2.3) and quality (section A.4) requirements. It is therefore mandatory to be able to assess the source code for
>
> - independent verification, and/or
> - performance profiling, and/or
> - quality control
>
> by qualified third parties from all around the world. Hence, it has to be *open source* according to the definition of the Open Source Initiative.
>
> On the other hand, the initial version of the tool is expected to provide a basic functionality which might be extended (section A.1.1 and section A.2.6) by academic researchers and/or professional programmers. It thus should also be *free*—in the sense of freedom, not in the sense of price—as defined by the Free Software Foundation. There is no requirement on the pricing scheme, which is up to the vendor to define in the offer along with the detailed licensing terms. These should allow users to solve their problems the way they need and, eventually, to modify and improve the tool to suit their needs. If they cannot program themselves, they should have the *freedom* to hire somebody to do it for them.

FeenoX is a cloud-first computational tool aimed at solving engineering problems with a particular design basis, as explained in

- [redacted] (2024). FeenoX: a cloud-first finite-element(ish) computational engineering tool. Journal of Open Source Software, 9(95), 5846. https://doi.org/10.21105/joss.05846

"Cloud first" vs. "cloud friendly"

In web design theory, there is a difference between mobile-first and mobile-friendly interfaces. In the same sense, FeenoX is cloud first and not just cloud friendly.

But what does this mean? Let us first start with the concept of "cloud friendliness," meaning that it is *possible* to run something on the cloud without substantial effort. This implies that a computational tool is cloud friendly if it

1. can be executed remotely without any special care, i.e. a GNU/Linux binary ran on a server through SSH,
2. can exploit the (in principle) unbounded resources provided by a set of networked servers, and
3. does not need interactive input meaning that, once launched, it can finish without needing further human intervention.

Yet, a cloud-first tool needs to take account other more profound concepts as well in early-stage design decisions. In software development, the modification of an existing desktop-based piece of software to allow remote execution is called "cloud-enabling." In words of a senior manager, "cloud development is the opposite of desktop development." So starting from scratch a cloud-first tool is a far better approach than refactoring an existing desktop program to make it cloud friendly.

For instance, to make proper use of the computational resources available in remote servers launched on demand, it is needed to

- have all the hosts in a particular network
- configure a proper domain name service
- design shared network file systems
- etc.

Instead of having to manually perform this set up each time a calculation is needed, one can automatize the workflow with *ad-hoc* scripts acting as "thin clients" which would, for instance,

- launch and configure the remote computing instances, optionally using containerization technology
- send the input files needed by the computational tools
- launch the actual computational tools (Gmsh, FeenoX, etc.) over the instances, e.g. using `mpiexec` or similar to be able to either
  a. to reduce the wall time needed to solve a problem, and/or
  b. to allow the solution of large problems that do not fit into a single computer
- monitor and communicate with the solver as the calculation progresses

- handle eventual errors
- get back and process the results

Furthermore, we could design and implement more complex clients able to handle things like

- authentication
- resource management (i.e. CPU hours)
- estimation of the number and type of instances needed to solve a certain problem
- parametric sweeps
- optimization loops
- conditionally-chained simulations
- etc.

Therefore, the computational tools that would perform the actual calculations should be designed in such a way not only to allow these kind of workflows but also to make them efficient. In fact, we say "clients" in plural because—as the Unix rule of diversity (section C.16) asks for–depending on the particular problem type and requirements different clients might be needed. And since FeenoX itself is flexible enough to be able to solve not only different types of partial differential equations but also

- different types of problems
  - coupled
  - parametric
  - optimization
  - etc.
- in different environments
  - many small cases
  - a few big ones
  - only one but huge
  - etc.
- under different conditions
  - in the industry by a single engineer
  - in the academy by several researchers
  - as a service in a public platform
  - etc.

then it is expected nor to exist a one-size-fits-all solution able to handle all the combinations in an optimum way.

However, if the underlying computational tool has been carefully designed to be able to handle all these details and to be flexible enough to accommodate other eventual and/or unexpected requirements by design, then we say that the tool is "cloud first." Throughout this SDS we thoroughly explain the features of this particular cloud-first design. Indeed, FeenoX is essentially a back end which can work with a number of different front ends (figure B.1), including these thin clients and web-based interfaces (figure B.2)

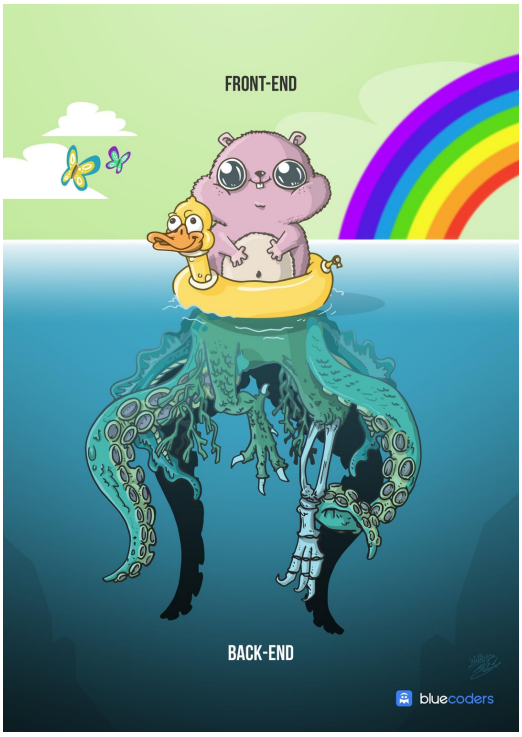


Figure B.1.: Conceptual illustration of the difference between a front end and a back end ©bluecoders.

Unfair advantage

To better illustrate FeenoX's unfair advantage (in the entrepreneurial sense), let us first consider what the options are when we need to write a technical report, paper or document:

Feature	Microsoft Word	Google Docs	Markdown[1]	(La)TeX
Aesthetics	×	×	✓	✓
Convertibility (to other formats)	∼	∼	✓	∼
Traceability	×	∼	✓	✓
Mobile-friendliness	×	✓	✓	×
Collaborativeness	×	✓	✓	∼
Licensing/openness	×	×	✓	✓
Non-nerd friendliness	✓	✓	∼	×

After analyzing the pros and cons of each alternative, at some point it should be evident that Markdown (plus friends) gives the best trade off. We can then perform a similar analysis for the options available in order to solve an engineering problem casted as a partial differential equation, say by using a finite-element formulation:

[1]Here "Markdown" means (Pandoc + Git + Github / Gitlab / Gitea)

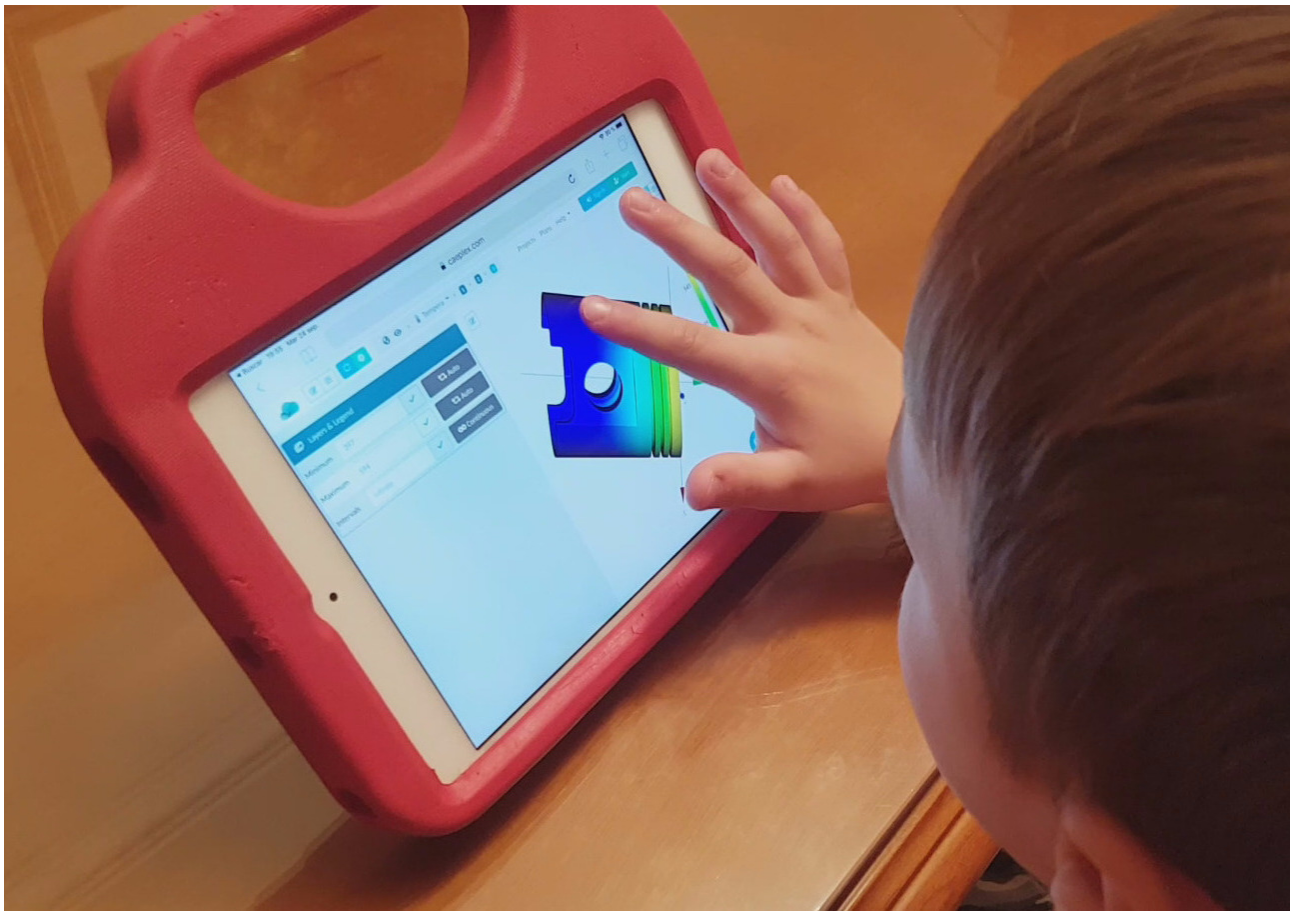


Figure B.2.: The web-based platform CAEplex for FeenoX. https://www.youtube.com/watch?v=7KqiMbrSLDc

Feature	Desktop GUIs	Web frontends	FeenoX[2]	Libraries
Flexibility	∼	×	✓	✓
Scalability	×	∼	✓	✓
Traceability	×	∼	✓	✓
Cloud-friendliness	×	✓	✓	✓
Collaborativeness	×	✓	✓	∼
Licensing/openness	✓/∼/×	×	✓	✓
Non-nerd friendliness	✓	✓	∼	×

Therefore, FeenoX is—in a certain sense—to desktop FEA programs like

- Code_Aster with Salome-Meca, or
- CalculiX with PrePoMax

and to libraries like

- MoFEM or
- Sparselizard

what Markdown is to Word and (La)TeX, respectively and *deliberately*.

Licensing

FeenoX is licensed under the terms of the GNU General Public License version 3 or, at the user convenience, any later version. This means that users get the four essential freedoms:[3]

0. The freedom to *run* the program as they wish, for *any* purpose.
1. The freedom to *study* how the program works, and *change* it so it does their computing as they wish.
2. The freedom to *redistribute* copies so they can help others.
3. The freedom to *distribute* copies of their *modified* versions to others.

So a free program has to be open source, but it also has to explicitly provide the four freedoms above both through the written license and through appropriate mechanisms to get, modify, compile, run and document these modifications using well-established and/or reasonable straightforward procedures. That is why licensing FeenoX as GPLv3+ also implies that the source code and all the scripts and makefiles needed to compile and run it are available for anyone that requires it (i.e. it is compiled with `./configure && make`). Anyone wanting to modify the program either to fix bugs, improve it or add new features is free to do so. And if they do not know how to program, the have

[2] Here "FeenoX" means (FeenoX + Gmsh + Paraview + Git + Github / Gitlab / Gitea)

[3] There are some examples of pieces of computational software which are described as "open source" in which even the first of the four freedoms is denied. The most iconic case is that of Android, whose sources are readily available online but there is no straightforward way of updating one's mobile phone firmware with a customized version, not to mention vendor and hardware lock ins and the possibility of bricking devices if something unexpected happens. In the nuclear industry, it is the case of a Monte Carlo particle-transport program that requests users to sign an agreement about the objective of its usage before allowing its execution. The software itself might be open source because the source code is provided after signing the agreement, but it is not free (as in freedom) at all.

the freedom to hire a programmer to do it without needing to ask permission to the original authors. Even more, the documentation is released under the terms of the GNU Free Documentation License so these new (or modified) features can be properly documented as well.

Nevertheless, since these original authors are the copyright holders, they still can use it to either enforce or prevent further actions from the users that receive FeenoX under the GPLv3+. In particular, the license allows re-distribution of modified versions only if

a. they are clearly marked as different from the original, and
b. they are distributed under the same terms of the GPLv3+.

There are also some other subtle technicalities that need not be discussed here such as

- what constitutes a modified version (which cannot be redistributed under a different license)
- what is an aggregate (in which each part be distributed under different licenses)
- usage over a network and the possibility of using AGPL instead of GPL to further enforce freedom

These issues are already taken into account in the FeenoX licensing scheme.

It should be noted that not only is FeenoX free and open source, but also all of the libraries it depends on (and their dependencies) also are. It can also be compiled using free and open source build tool chains running over free and open source operating systems.

To sum up this introduction, FeenoX is…

1. a cloud-first computational tool (not just cloud *friendly,* but cloud **first**).
2. to traditional computational software and to specialized libraries what Markdown is to Word and TeX, respectively.
3. both free (as in freedom) and open source.

B.1.1. Objective

> The main objective of the tool is to be able to solve engineering problems which are usually casted as differential-algebraic equations (DAEs) or partial differential equations (PDEs), such as
>
> - heat conduction
> - mechanical elasticity
> - structural modal analysis
> - mechanical frequency studies
> - electromagnetism
> - chemical diffusion
> - process control dynamics
> - computational fluid dynamics
> - …
>
> on one or more mainstream cloud servers, i.e. computers with hardware and operating systems (further discussed in section A.2) that allows them to be available online and accessed remotely

> either interactively or automatically by other computers as well. Other architectures such as high-end desktop personal computers or even low-end laptops might be supported but they should not the main target (i.e. the tool has to be cloud-first but laptop-friendly).
>
> The initial version of the tool must be able to handle a subset of the above list of problem types. Afterward, the set of supported problem types, models, equations and features of the tool should grow to include other models as well, as required in section A.2.6.

The choice of the initial supported features is based on the types of problem that the FeenoX's precursor codes (namely wasora, Fino and milonga, referred to as "previous versions" from now on) already have been supporting since more than ten years now. A subsequent road map and release plans can be designed as requested. FeenoX's first version includes a subset of the required functionality, namely

- open and closed-loop dynamical systems
- Laplace/Poisson/Helmholtz equations
- heat conduction
- mechanical elasticity
- structural modal analysis
- multi-group neutron transport and diffusion

Sección B.2.6 explains the mechanisms that FeenoX provides in order to add (or even remove) other types of problems to be solved.

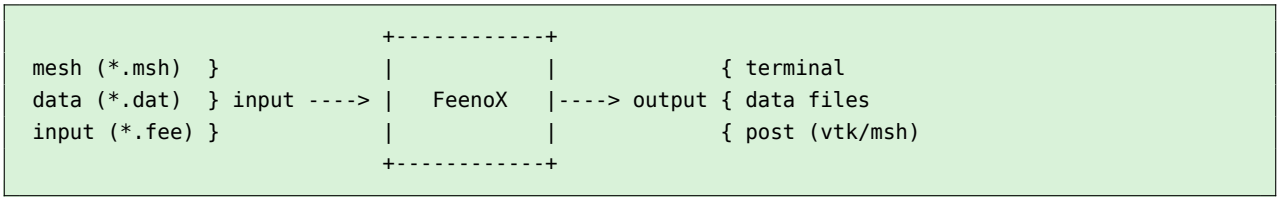
Recalling that FeenoX is a "cloud-first" tool as explained in section B.1, it is designed to be developed and executed primarily on GNU/Linux hosts, which is the architecture of more than 90% of the internet servers which we collectively call "the public cloud." It should be noted that GNU/Linux is a POSIX-compliant operating system which is compatible with Unix, and that FeenoX was designed and implemented following the rules of Unix philosophy which is further explained in section C. Besides the POSIX standard, as explained below in section B.2.4, FeenoX also uses MPI which is a well-known industry standard for massive execution of parallel processes following the distributed-systems parallelization paradigm. Finally, if performance and/or scalability are not important issues, FeenoX can be run in a (properly cooled) local PC, laptop or even in embedded systems such as Raspberry Pi (see section B.2).

B.1.2. Scope

> The tool should allow users to define the problem to be solved programmatically. That is to say, the problem should be completely defined using one or more files either…
>
> a. specifically formatted for the tool to read such as JSON or a particular input format (historically called input decks in punched-card days), and/or
> b. written in an high-level interpreted language such as Python or Julia.
>
> Once the problem has been defined and passed on to the solver, no further human intervention should be required.
>
> It should be noted that a graphical user interface is *not* required. The tool may include one,

> but it should be able to run without needing any interactive user intervention rather than the preparation of a set of input files. Nevertheless, the tool might *allow* a GUI to be used. For example, for a basic usage involving simple cases, a user interface engine should be able to create these problem-definition files in order to give access to less advanced users to the tool using a desktop, mobile and/or web-based interface in order to run the actual tool without needing to manually prepare the actual input files.
>
> However, for general usage, users should be able to completely define the problem (or set of problems, i.e. a parametric study) they want to solve in one or more input files and to obtain one or more output files containing the desired results, either a set of scalar outputs (such as maximum stresses or mean temperatures), and/or a detailed time and/or spatial distribution. If needed, a discretization of the domain may to be taken as a known input, i.e. the tool is not required to create the mesh as long as a suitable mesher can be employed using a similar workflow as the one specified in this SRS.
>
> The tool should define and document (section A.4.6) the way the input files for a solving particular problem are to be prepared (section A.3.1) and how the results are to be written (section A.3.2). Any GUI, pre-processor, post-processor or other related graphical tool used to provide a graphical interface for the user should integrate in the workflow described in the preceding paragraph: a pre-processor should create the input files needed for the tool and a post-processor should read the output files created by the tool.

Since FeenoX is designed to be executed *in the cloud*, it works very much like a transfer function between one (or more) files and zero or more output files:



Technically speaking, FeenoX can be seen as a Unix filter designed to read an ASCII-based stream of characters (i.e. the input file, which in turn can include other input files or contain instructions to read data from mesh and/or other data files) and to write ASCII-formatted data into the standard output and/or other files. The input file can be prepared either by a human or by another program. The output stream and/or files can be read by either a human and/or another programs. A quotation from Eric Raymond's The Art of Unix Programming helps to illustrate this idea:

> Doug McIlroy, the inventor of Unix pipes and one of the founders of the Unix tradition, had this to say at the time:
>
> (i) Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new features.
>
> (ii) Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.
>
> [...]

> He later summarized it this way (quoted in "A Quarter Century of Unix" in 1994):
>
> - This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

Keep in mind that even though both the quotes above and many finite-element programs that are still mainstream today date both from the early 1970s, fifty years later the latter still

- do not make just only one thing well,
- do complicate old programs by adding new features,
- do not expect their output to become the input to another,
- do clutter output with extraneous information,
- do use stringently columnar and/or binary input (and output!) formats, and/or
- do insist on interactive input.

There are other FEA tools that, even though born closer in time, also follow the above bullets literally. But FeenoX does not, since it follows the Unix philosophy in general and Eric Raymond's 17 Unix Rules (section C) in particular. One of the main ideas is the *rule of separation* (section C.4) that essentially asks to separate mechanism from policy, that in the computational engineering world translates into separating the front end from the back end as illustrated in figure B.1.

When solving ordinary differential equations, the usual workflow involves solving them with FeenoX and plotting the results with Gnuplot or Pyxplot. When solving partial differential equations (PDEs), the mesh is created with Gmsh and the output can be post-processed with Gmsh, Paraview or any other post-processing system (even a web-based interface) that follows rule of separation. Even though most FEA programs eventually separate the interface from the solver up to some degree, there are cases in which they are still dependent such that changing the former needs updating the latter. This is the usual case with legacy programs designed back in the 1990s (or even one or two decades before) that are still around nowadays. They usually still fulfill almost all of the bullets above and are the ones which their owners are trying to convert from desktop to cloud-enabled programs instead of starting from scratch.

From the very beginning, FeenoX is designed as a pure back end which should nevertheless provide appropriate mechanisms for different front ends to be able to communicate and to provide a friendly interface for the final user. Yet, the separation is complete in the sense that the nature of the front ends can radically change (say from a desktop-based point-and-click program to a web-based interface or an immersive augmented-reality application with goggles) without needing the modify the back end. Not only far more flexibility is given by following this path, but also develop efficiency and quality is encouraged since programmers working on the lower-level of an engineering tool usually do not have the skills needed to write good user-experience interfaces, and conversely.

In the very same sense, FeenoX does not discretize continuous domains for PDE problems itself, but relies on separate tools for this end. Fortunately, there already exists one meshing tool which is FOSS (GPLv2) and shares most (if not all) of the design basis principles with FeenoX: the three-dimensional finite element mesh generator Gmsh.

Strictly speaking, FeenoX does not need to be used along with Gmsh but with any other mesher able to write meshes in Gmsh's format `.msh`. But since Gmsh also

- is free and open source,
- works also in a transfer-function-like fashion,
- runs natively on GNU/Linux,
- has a similar (but more comprehensive) API for Python/Julia,
- etc.

it is a perfect match for FeenoX. Even more, it provides suitable domain decomposition methods (through other open-source third-party libraries such as Metis) for scaling up large problems.

B.1.2.1. NAFEMS LE10 benchmark

Let us solve the linear elasticity benchmark problem NAFEMS LE10 "Thick plate pressure." with FeenoX. Note the one-to-one correspondence between the human-friendly problem statement from figure B.3 and the FeenoX input file:

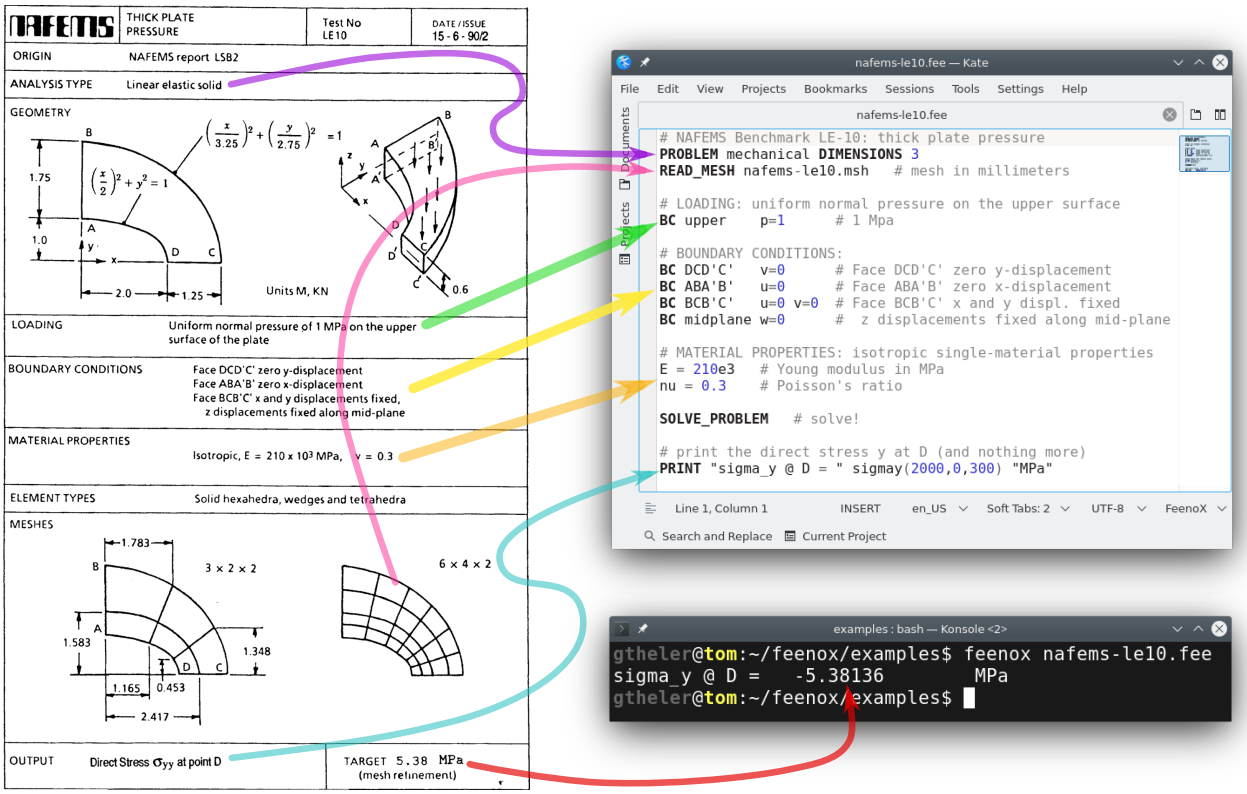


Figure B.3.: The NAFEMS LE10 problem statement and the corresponding FeenoX input

```
# NAFEMS Benchmark LE-10: thick plate pressure
PROBLEM mechanical DIMENSIONS 3
READ_MESH nafems-le10.msh   # mesh in millimeters

# LOADING: uniform normal pressure on the upper surface
BC upper    p=1       # 1 Mpa

# BOUNDARY CONDITIONS:
BC DCD'C'   v=0       # Face DCD'C' zero y-displacement
BC ABA'B'   u=0       # Face ABA'B' zero x-displacement
BC BCB'C'   u=0 v=0   # Face BCB'C' x and y displ. fixed
```

```
BC midplane w=0      #  z displacements fixed along mid-plane

# MATERIAL PROPERTIES: isotropic single-material properties
E = 210e3   # Young modulus in MPa
nu = 0.3    # Poisson's ratio

SOLVE_PROBLEM   # solve!

# print the direct stress y at D (and nothing more)
PRINT "σ_y @ D = " sigmay(2000,0,300) "MPa"
```

Here, "one-to-one" means that the input file does not need any extra definition which is not part of the problem formulation. Of course the cognizant engineer *can* give further definitions such as

- the linear solver and pre-conditioner
- the tolerances for iterative solvers
- options for computing stresses out of displacements
- etc.

However, she *is not obliged to* as–at least for simple problems—the defaults are reasonable. This is akin to writing a text in Markdown where one does not need to care if the page is A4 or letter (as, in most cases, the output will not be printed but rendered in a web browser).

The problem asks for the normal stress in the y direction σ_y at point "D," which is what FeenoX writes (and nothing else, *rule of economy*):

```
$ feenox nafems-le10.fee
sigma_y @ D =   -5.38016        MPa
$
```

Also note that since there is only one material, there is no need to do an explicit link between material properties and physical volumes in the mesh (*rule of simplicity*). And since the properties are uniform and isotropic, a single global scalar for E and a global single scalar for ν are enough.

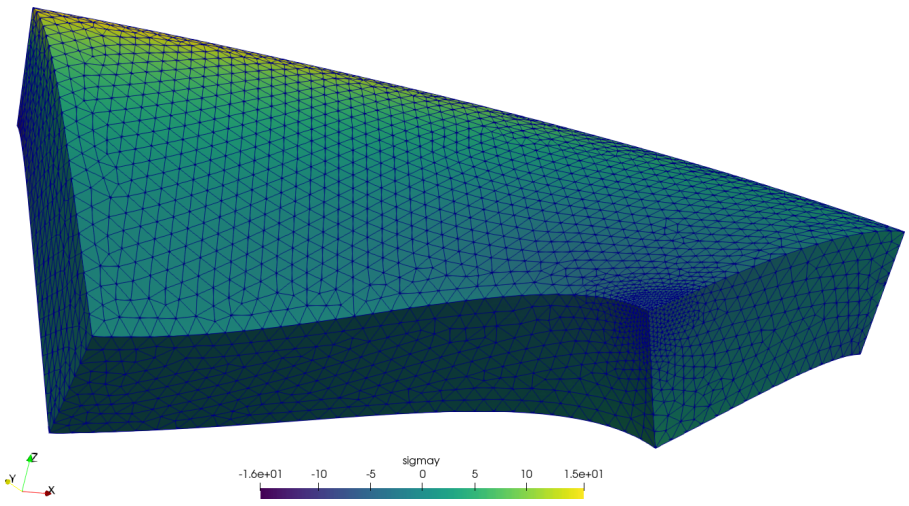


Figure B.4.: Normal stress σ_y refined around point D over 5,000x-warped displacements for LE10 created with Paraview

For the sake of visual completeness, post-processing data with the scalar distribution of σ_y and the vector field of displacements $[u, v, w]$ can be created by adding one line to the input file:

```
WRITE_MESH nafems-le10.vtk sigmay VECTOR u v w
```

This VTK file can then be post-processed to create interactive 3D views, still screenshots, browser and mobile-friendly webGL models, etc. In particular, using Paraview one can get a colorful bitmapped PNG (the displacements are far more interesting than the stresses in this problem).

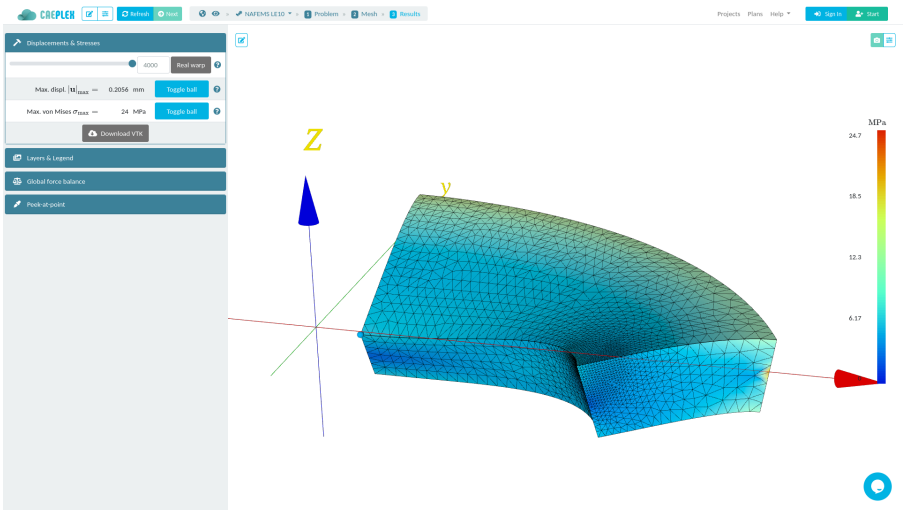


Figure B.5.: See also https://caeplex.com/r/f1a82f to see this very same LE10 problem solved in the mobile-friendly web-based interface CAEplex that uses FeenoX as the back end

B.1.2.2. The Lorenz chaotic system

Let us consider the famous chaotic Lorenz's dynamical system. Here is one way of getting an image of the butterfly-shaped attractor using FeenoX to compute it and Gnuplot to draw it. Solve

$$\begin{cases} \dot{x} &= \sigma \cdot (y - x) \\ \dot{y} &= x \cdot (r - z) - y \\ \dot{z} &= xy - bz \end{cases}$$

for $0 < t < 40$ with initial conditions

$$\begin{cases} x(0) = -11 \\ y(0) = -16 \\ z(0) = 22.5 \end{cases}$$

and $\sigma = 10$, $r = 28$ and $b = 8/3$, which are the classical parameters that generate the butterfly as presented by Edward Lorenz back in his seminal 1963 paper Deterministic non-periodic flow.

The following ASCII input file resembles the parameters, initial conditions and differential equations of the problem as naturally as possible:

```
PHASE_SPACE x y z      # Lorenz 'attractors phase space is x-y-z
end_time = 40          # we go from t=0 to 40 non-dimensional units

sigma = 10             # the original parameters from the 1963 paper
r = 28
b = 8/3

x_0 = -11              # initial conditions
y_0 = -16
z_0 = 22.5

# the dynamical system's equations written as naturally as possible
x_dot = sigma*(y - x)
y_dot = x*(r - z) - y
z_dot = x*y - b*z

PRINT t x y z         # four-column plain-ASCII output
```

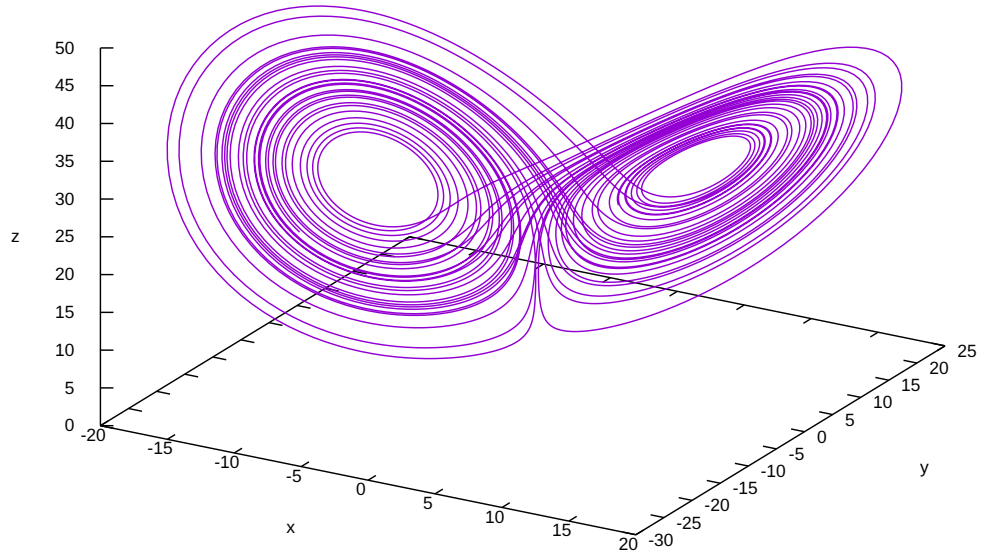


Figure B.6.: The Lorenz attractor solved with FeenoX and drawn with Gnuplot

Indeed, when executing FeenoX with this input file, we get four ASCII columns (t, x, y and z) which we can then redirect to a file and plot it with a standard tool such as Gnuplot. Note the importance of relying on plain ASCII text formats both for input and output, as recommended by the Unix philosophy and the *rule of composition*: other programs can easily create inputs for FeenoX and other programs can easily understand FeenoX's outputs. This is essentially how Unix filters and pipes work.

Note the one-to-one correspondence between the human-friendly differential equations (written in TeX and rendered as typesetted mathematical symbols) and the computer-friendly input file that FeenoX reads.

Even though the initial version of FeenoX does not provide an API for high-level interpreted languages such as Python or Julia, the code is written in such a way that this feature can be added without needing a major refactoring. This will allow to fully define a problem in a procedural way, increasing also flexibility.

B.2. Architecture

> The tool must be aimed at being executed unattended on remote servers which are expected to have a mainstream (as of the 2020s) architecture regarding operating system (GNU/Linux variants and other Unix-like OSes) and hardware stack, such as
>
> - a few Intel-compatible or ARM-like CPUs per host
> - a few levels of memory caches
> - a few gigabytes of random-access memory
> - several gigabytes of solid-state storage
>
> It should successfully run on
>
> - bare-metal
> - virtual servers
> - containerized images
>
> using standard compilers, dependencies and libraries already available in the repositories of most current operating systems distributions.
>
> Preference should be given to open source compilers, dependencies and libraries. Small problems might be executed in a single host but large problems ought to be split through several server instances depending on the processing and memory requirements. The computational implementation should adhere to open and well-established parallelization standards.
>
> Ability to run on local desktop personal computers and/laptops is not required but suggested as a mean of giving the opportunity to users to test and debug small coarse computational models before launching the large computation on a HPC cluster or on a set of scalable cloud instances. Support for non-GNU/Linux operating systems is not required but also suggested.
>
> Mobile platforms such as tablets and phones are not suitable to run engineering simulations due to their lack of proper electronic cooling mechanisms. They are suggested to be used to control one (or more) instances of the tool running on the cloud, and even to pre and post process results through mobile and/or web interfaces.

Very much like the C language (after A & B) and Unix itself (after a first attempt and the failed MULTICS), FeenoX can be seen as a third-system effect:

> A notorious 'second-system effect' often afflicts the successors of small experimental prototypes. The urge to add everything that was left out the first time around all too frequently leads to huge and overcomplicated design. Less well known, because less common, is the 'third-system effect': sometimes, after the second system has collapsed of its own weight, there is a chance to go back to simplicity and get it right.
>
> From Eric Raymond's The Art of Unix Programming

Feenox is indeed the third version written from scratch after a first implementation in 2009 (different small components with different names) and a second one (named wasora that allowed dynamically-shared plugins to be linked at runtime to provide particular PDEs) which was far more complex and

had far more features circa 2012–2015. The third attempt, FeenoX, explicitly addresses the "do one thing well" idea from Unix.

Furthermore, not only is FeenoX itself both free and open-source software but, following the *rule of composition* (section C.3), it also is designed to connect and to work with other free and open source software such as

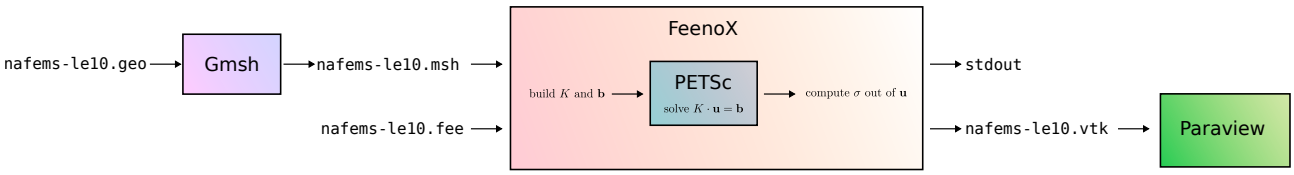
- Gmsh for pre and/or post-processing
- ParaView for post-processing
- Gnuplot for plotting 1D/2D results
- Pyxplot for plotting 1D results
- Pandoc for creating tables and documents
- TeX for creating tables and documents

and many others, which are readily available in all major GNU/Linux distributions.

FeenoX also makes use of high-quality free and open source mathematical libraries which contain numerical methods designed by mathematicians and implemented by professional programmers. In particular, it depends on

- GNU Scientific Library for general mathematics,
- SUNDIALS IDA for ODEs and DAEs,
- PETSc for linear, non-linear and transient PDEs, and
- SLEPc for PDEs involving eigen problems

Therefore, if one zooms in into the block of the transfer function above, FeenoX can also be seen as a glue layer between the input files defining a physical problem and the mathematical libraries used to solve the discretized equations. For example, when solving the linear elastic problem from the NAFEMS LE10 case discussed above, we can draw the following diagram:



This way, FeenoX bounds its scope to do only one thing and to do it well: to build and solve finite-element formulations of physical problems. And it does so on high grounds, both ethical and technological:

a. Ethical, since it is free software, all users can

   0. run,
   1. share,
   2. modify, and/or
   3. re-share their modifications.

   If a user cannot read or write code to make FeenoX suit her needs, at least she has the *freedom* to hire someone to do it for her.

b. Technological, since it is open source, advanced users can detect and correct bugs and even improve the algorithms. Given enough eyeballs, all bugs are shallow.

FeenoX's main development architecture is Debian GNU/Linux running over 64-bits Intel-compatible processors (but binaries for ARM architectures can be compiled as well). All the dependencies are free and/or open source and already available in Debian's latest stable official repositories, as explained in section B.2.1.

The POSIX standard is followed whenever possible, allowing thus FeenoX to be compiled in other operating systems and architectures such as Windows (using Cygwin) and MacOS. The build procedure is the well-known and mature `./configure && make` command.

FeenoX is written in C conforming to the ISO C99 specification (plus POSIX extensions), which is a standard, mature and widely supported language with compilers for a wide variety of architectures. As listed above, for its basic mathematical capabilities, FeenoX uses the GNU Scientific Library. For solving ODEs/DAEs, FeenoX relies on Lawrence Livermore's SUNDIALS library. For PDEs, FeenoX uses Argonne's PETSc library and Universitat Politècnica de València's SLEPc library. All of them are

- free and open source,
- written in C (neither Fortran nor C++),
- mature and stable,
- actively developed and updated,
- very well known both in the industry and academia.

Moreover, PETSc and SLEPc are scalable through the MPI standard, further discussed in section B.2.4. This means that programs using both these libraries can run on either large high-performance supercomputers or low-end laptops. FeenoX has been run on

- Raspberry Pi
- Laptop (GNU/Linux & Windows 10)
- Macbook
- Desktop PC
- Bare-metal servers
- Vagrant/Virtualbox virtual machines
- Docker/Kubernetes containers
- AWS/DigitalOcean/Contabo instances

Due to the way that FeenoX is designed and the policy separated from the mechanism, it is possible to control a running instance remotely from a separate client which can eventually run on a mobile device ([figure B.2,figure B.5]).

The following example illustrates how well FeenoX works as one of many links in a chain that goes from tracing a bitmap with the problem's geometry down to creating a nice figure with the results of a computation.

Say you are Homer J. Simpson and you want to solve a maze drawn in a restaurant's placemat while driving to your wife's aunt funeral. One where both the start and end points are known beforehand as show in figure B.7. In order to avoid falling into the alligator's mouth, you can exploit the ellipticity of the Laplacian operator to solve any maze (even a hand-drawn one) without needing any fancy AI or ML algorithm. Just FeenoX and a bunch of standard open source tools to convert a bitmapped picture of the maze into an unstructured mesh.

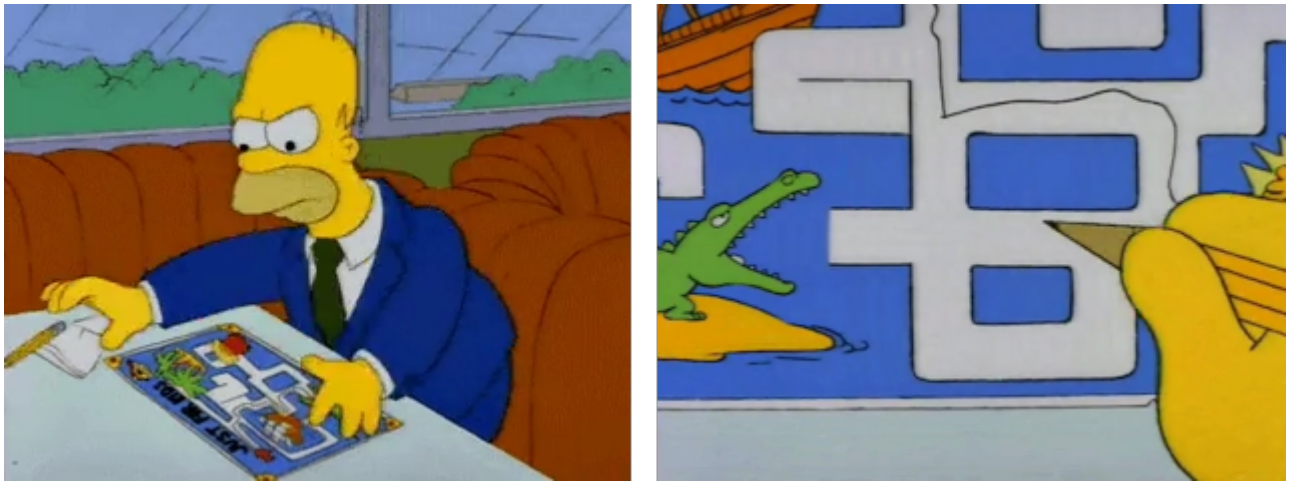
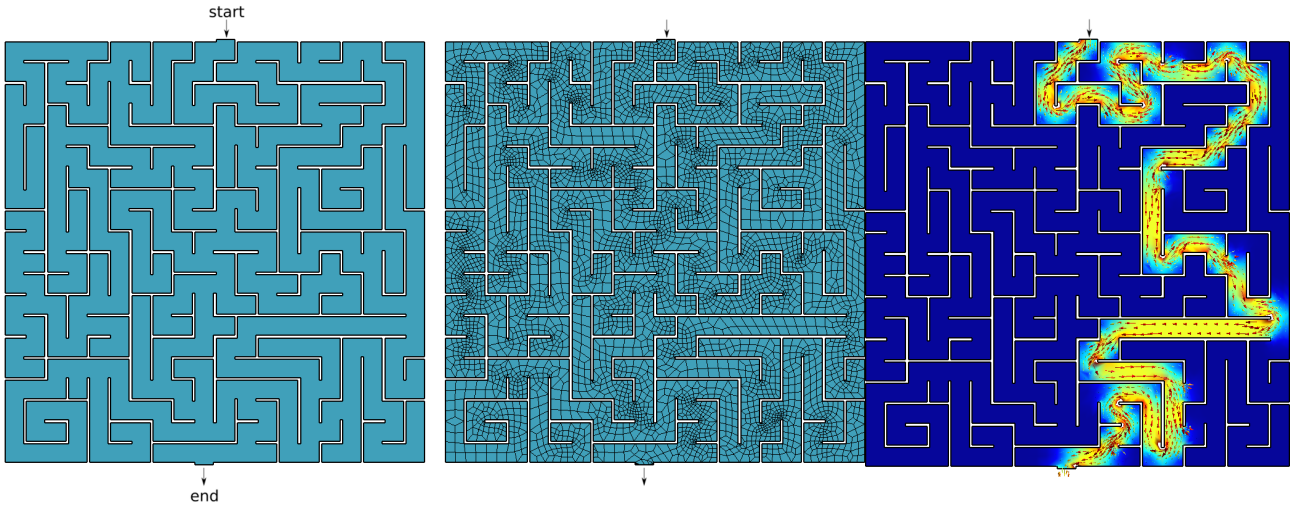


Figure B.7.: Homer trying to solve a maze on a placemat during season four.



(a) Bitmapped maze from https://www.mazegenerator.net (left) and 2D mesh (right)

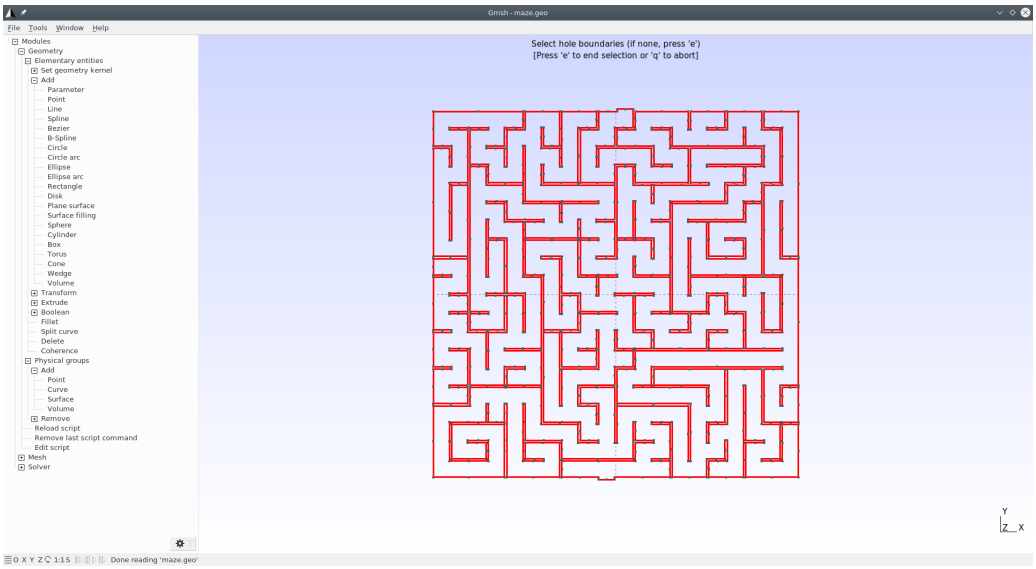
(b) Solution to found by FeenoX (and drawn by Gmsh)

Figure B.8.: Bitmapped, meshed and solved mazes.

1. Go to http://www.mazegenerator.net/
2. Create a maze
3. Download it in PNG (figure B.8a)
4. Perform some conversions
   - PNG → PNM → SVG → DXF → GEO

```
$ wget http://www.mazegenerator.net/static/orthogonal_maze_with_20_by_20_cells.png
$ convert orthogonal_maze_with_20_by_20_cells.png -negate maze.png
$ potrace maze.pnm --alphamax 0  --opttolerance 0 -b -o maze
$ .2dxf maze maze.dxf
$ ./dxf2geo maze.dxf 0.1
```

5. Open it with Gmsh



   - Add a surface
   - Set physical curves for “start” and “end”

6. Mesh it (figure B.8a)

```
gmsh -2 maze.geo
```

7. Solve $\nabla^2\phi = 0$ with BCs

$$\begin{cases} \phi = 0 & \text{at “start”} \\ \phi = 1 & \text{at “end”} \\ \nabla\phi \cdot \hat{\mathbf{n}} = 0 & \text{everywhere else} \end{cases}$$

```
PROBLEM laplace 2D  # pretty self-descriptive, isn't it?
READ_MESH maze.msh

# boundary conditions (default is homogeneous Neumann)
BC start  phi=0
BC end    phi=1

SOLVE_PROBLEM

# write the norm of gradient as a scalar field
# and the gradient as a 2d vector into a .msh file
WRITE_MESH maze-solved.msh \
    sqrt(dphidx(x,y)^2+dphidy(x,y)^2) \
    VECTOR dphidx dphidy 0
```

```
$ feenox maze.fee
$
```

8. Open `maze-solved.msh`, go to start and follow the gradient $\nabla\phi$!

B.2.1. Deployment

> The tool should be easily deployed to production servers. Both
>
> a. an automated method for compiling the sources from scratch aiming at obtaining optimized binaries for a particular host architecture should be provided using a well-established procedures, and
> b. one (or more) generic binary version aiming at common server architectures should be provided.
>
> Either option should be available to be downloaded from suitable online sources, either by real people and/or automated deployment scripts.

As already stated, FeenoX can be compiled from its sources using the well-established `configure` & `make` procedure. The code's source tree is hosted on Github so cloning the repository is the preferred way to obtain FeenoX, but source tarballs are periodically released too according to the requirements in section B.4.1. There are also non-official binary `.deb` packages which can be installed with `apt` using a custom package repository location.

The configuration and compilation is based on GNU Autotools that has more than thirty years of maturity and it is the most portable way of compiling C code in a wide variety of Unix variants. It has been tested with

- GNU C compiler (free)
- LLVM Clang compiler (free)
- Intel oneAPI C compiler (privative)

FeenoX depends on the four open source libraries stated in section B.2, although the last three of them are optional. The only mandatory library is the GNU Scientific Library which is part of the

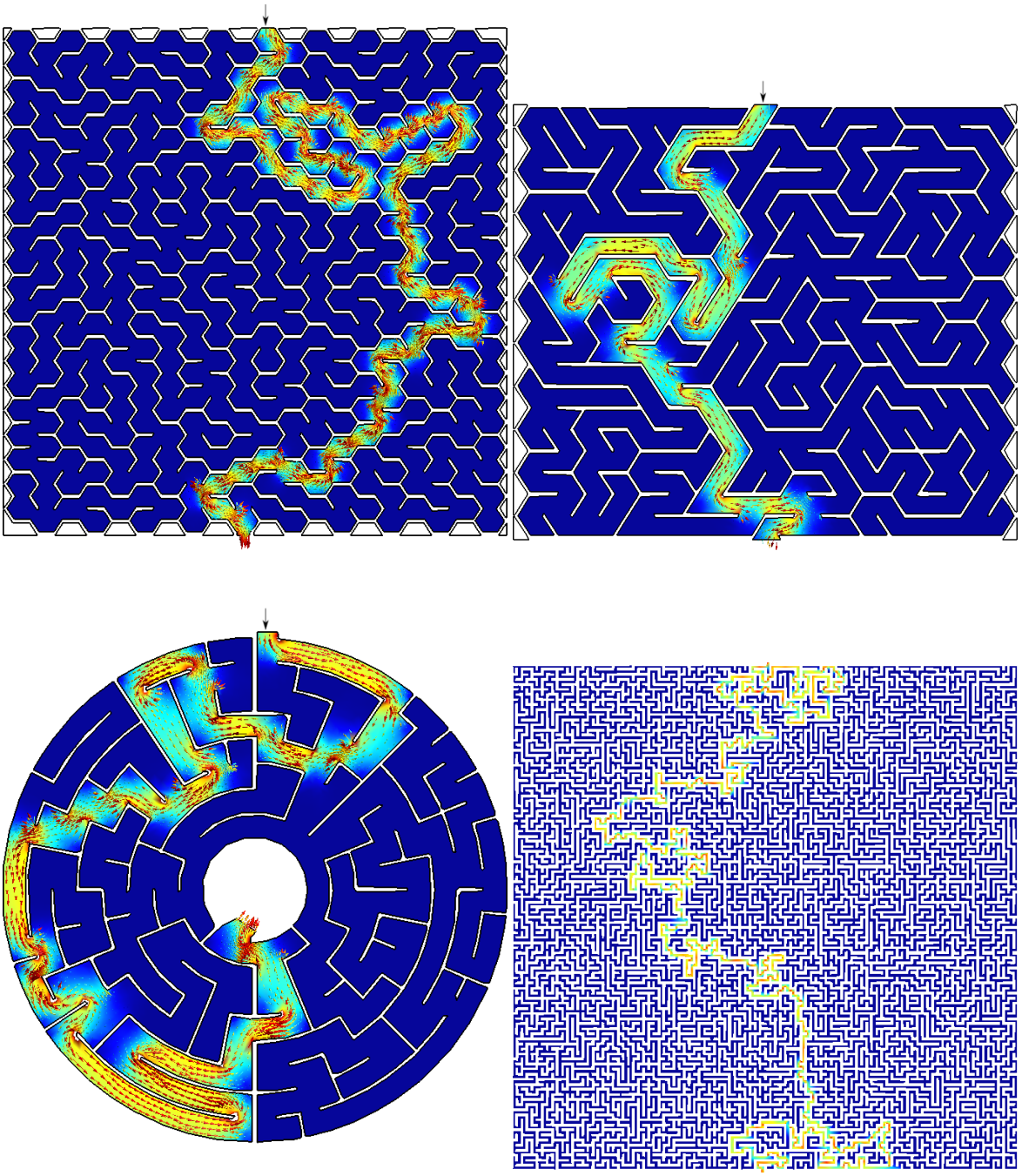


Figure B.9.: Any arbitrary maze (even hand-drawn) can be solved with FeenoX

GNU/Linux operating system and as such is readily available in all distributions as `libgsl-dev`. The sources of the rest of the optional libraries are also widely available in most common GNU/Linux distributions.

In effect, doing

```
sudo apt-get install gcc make libgsl-dev libsundials-dev petsc-dev slepc-dev
```

is enough to provision all the dependencies needed compile FeenoX from the source tarball with the full set of features. If using the Git repository as a source, then Git itself and the GNU Autoconf and Automake packages are also needed:

```
sudo apt-get install git autoconf automake
```

Even though compiling FeenoX from sources is the recommended way to obtain the tool—since the target binary can be compiled using particularly suited compilation options, flags and optimizations (especially those related to MPI, linear algebra kernels and direct and/or iterative sparse solvers)–there are also tarballs and `.deb` packages with usable binaries for some of the most common architectures—including some non-GNU/Linux variants. These binary distributions contain statically-linked executable files that do not need any other shared libraries to be installed on the target host. However, their flexibility and efficiency is generic and far from ideal. Yet the flexibility of having an execution-ready distribution package for users that do not know how to compile C source code outweighs the limited functionality and scalability of the tool.

For example, first PETSc can be built with a `-Ofast` flag:

```
$ cd $PETSC_DIR
$ export PETSC_ARCH=linux-fast
$ ./configure --with-debug=0 COPTFLAGS="-Ofast"
$ make -j8
$ cd $HOME
```

And then not only can FeenoX be configured to use that particular PETSc build but also to use a different compiler such as Clang instead of GNU GCC and to use the same `-Ofast` flag to compile FeenoX itself:

```
$ git clone https://github.com/seamplex/feenox
$ cd feenox
$ ./autogen.sh
$ export PETSC_ARCH=linux-fast
$ ./configure MPICH_CC=clang CFLAGS=-Ofast
$ make -j8
# make install
```

If one does not care about the details of the compilation, then a pre-compiled statically-linked binary can be directly downloaded very much as when downloading Gmsh:

```
$ wget http://gmsh.info/bin/Linux/gmsh-Linux64.tgz
$ wget https://seamplex.com/feenox/dist/linux/feenox-linux-amd64.tar.gz
```

The full online documentation contains a compilation guide with further detailed explanations of each of the steps involved.

All the commands needed to either download a binary executable or to compile from source with customized optimization flags can be automatized. The repository contains a subdirectory `dist` with instructions and scripts to build

- source tarballs
- binary tarballs
- Debian-compatible `.deb` packages

This way, deployment of the solver can be customized and tweaked as needed, including creating Docker containers with a working version of FeenoX.

B.2.2. Execution

> It is mandatory to be able to execute the tool remotely, either with a direct action from the user or from a high-level workflow which could be triggered by a human or by an automated script. Since it is required for the tool to be able to be run distributed among different servers, proper means to perform this kind of remote executions should be provided. The calling party should be able to monitor the status during run time and get the returned error level after finishing the execution.
>
> The tool shall provide means to perform parametric computations by varying one or more problem parameters in a certain prescribed way such that it can be used as an inner solver for an outer-loop optimization tool. In this regard, it is desirable that the tool could compute scalar values such that the figure of merit being optimized (maximum temperature, total weight, total heat flux, minimum natural frequency, maximum displacement, maximum von Mises stress, etc.) is already available without needing further post-processing.

As requested by the SRS and explained in section B.1.2, FeenoX is a program that reads the problem to be solved at run-time and not a library that has to be linked against code that defines the problem. Since FeenoX is designed to run as

- a Unix filter, or
- as a transfer function between input and output files

and it explicitly avoids having a graphical interface, the binary executable works as any other Unix terminal command. Moreover, as discussed in section B.2.4, FeenoX uses the MPI standard for parallelization among several hosts. Therefore, it can be launched through the command `mpiexec` (or `mpirun`).

When invoked without arguments, it prints its version (a thorough explanation of the versioning scheme is given in section B.4.1), a one-line description and the usage options:

```
$ feenox
FeenoX v1.0.8-g731ca5d
a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool
```

```
usage: feenox [options] inputfile [replacement arguments] [petsc options]

  -h, --help         display options and detailed explanations of commmand-line usage
  -v, --version      display brief version information and exit
  -V, --versions     display detailed version information
  -c, --check        validates if the input file is sane or not
  --pdes             list the types of PROBLEMs that FeenoX can solve, one per line
  --elements_info    output a document with information about the supported element types
  --linear           force FeenoX to solve the PDE problem as linear
  --non-linear       force FeenoX to solve the PDE problem as non-linear

Run with --help for further explanations.
$
```

The program can also be executed remotely either

a. on a running server through a SSH session
   - in serial directly invoking the `feenox` binary
   - in parallel through the `mpiexec` wrapper, e.g.

     ```
     mpiexec -n 4 feenox input.fee
     ```

b. spawned by a daemon listening to a network requests,
c. in a container as part of a provisioning script,
d. in many other ways.

As explained in the help message, FeenoX can read the input from the standard input if `-` is specified as the input path. This is useful in scripts where small calculations are needed, e.g.

```
$ a=3
$ echo "PRINT 1/$a" | feenox -
0.333333
$
```

FeenoX provides mechanisms to inform its progress by writing certain information to devices or files, which in turn can be monitored remotely or even trigger server actions. Progress can be as simple as an ASCII bar (triggered with `--progress` in the command line or with the keyword `PROGRESS` in the input file) to more complex mechanisms like writing the status in a shared memory segment. Figura B.10 shows how the CAEplex platform shows the progress interactively in its web-based interface.

Regarding its execution, there are three ways of solving problems:

1. direct execution
2. parametric runs, and
3. optimization loops.

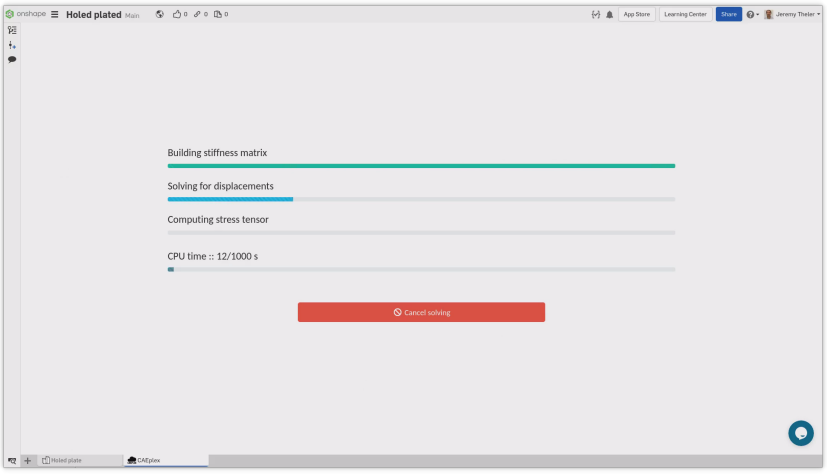


Figure B.10.: ASCII progress bars parsed and converted into a web-based interface

B.2.2.1. Direct execution

When directly executing FeenoX, one gives a single argument to the executable with the path to the main input file. For example, the following input computes the first twenty numbers of the Fibonacci sequence using the closed-form formula

$$f(n) = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}}$$

where $\varphi = (1+\sqrt{5})/2$ is the Golden ratio:

```
# the Fibonacci sequence using the closed-form formula as a function
phi = (1+sqrt(5))/2
f(n) = (phi^n - (1-phi)^n)/sqrt(5)
PRINT_FUNCTION f MIN 1 MAX 20 STEP 1
```

FeenoX can be directly executed to print the function $f(n)$ for $n = 1, \ldots, 20$ both to the standard output and to a file named `one` (because it is the first way of solving Fibonacci with Feenox):

```
$ feenox fibo_formula.fee | tee one
1       1
2       1
3       2
4       3
5       5
6       8
7       13
8       21
9       34
10      55
11      89
12      144
13      233
14      377
15      610
```

```
16      987
17      1597
18      2584
19      4181
20      6765
$
```

Now, we could also have computed these twenty numbers by using the direct definition of the sequence into a vector **f** of size 20. This time we redirect the output to a file named `two`:

```
# the fibonacci sequence as a vector
VECTOR f SIZE 20

f[i]<1:2> = 1
f[i]<3:vecsize(f)> = f[i-2] + f[i-1]

PRINT_VECTOR i f
```

```
$ feenox fibo_vector.fee > two
$
```

Finally, we print the sequence as an iterative problem and check that the three outputs are the same:

```
# the fibonacci sequence as an iterative problem

static_steps = 20
#static_iterations = 1476  # limit of doubles

IF step_static=1|step_static=2
 f_n = 1
 f_nminus1 = 1
 f_nminus2 = 1
ELSE
 f_n = f_nminus1 + f_nminus2
 f_nminus2 = f_nminus1
 f_nminus1 = f_n
ENDIF

PRINT step_static f_n
```

```
$ feenox fibo_iterative.fee > three
$ diff one two
$ diff two three
$
```

These three calls were examples of direct execution of FeenoX: a single call with a single argument to solve a single fixed problem.

B.2.2.2. Parametric

To use FeenoX in a parametric run, one has to successively call the executable passing the main input file path in the first argument followed by an arbitrary number of parameters. These extra

parameters will be expanded as string literals `$1`, `$2`, etc. appearing in the input file. For example, if `hello.fee` is

```
PRINT "Hello $1!"
```

then

```
$ feenox hello.fee World
Hello World!
$ feenox hello.fee Universe
Hello Universe!
$
```

To have an actual parametric run, an external loop has to successively call FeenoX with the parametric arguments. For example, say this file `cantilever.fee` fixes the face called "left" and sets a load in the negative z direction of a mesh called `cantilever-$1-$2.msh`. The output is a single line containing the number of nodes of the mesh and the displacement in the vertical direction $w(500,0,0)$ at the center of the cantilever's free face:

```
PROBLEM elastic 3D
READ_MESH cantilever-$1-$2.msh   # in meters

E = 2.1e11          # Young modulus in Pascals
nu = 0.3            # Poisson's ratio

BC left   fixed
BC right  tz=-1e5  # traction in Pascals, negative z

SOLVE_PROBLEM

# z-displacement (components are u,v,w) at the tip vs. number of nodes
PRINT nodes w(500,0,0) "\# $1 $2"
```

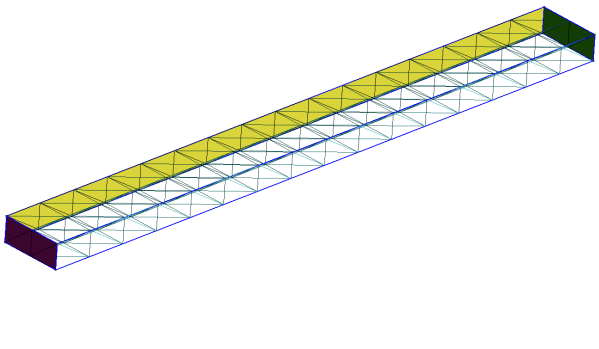


Figure B.11.: Tetrahedra

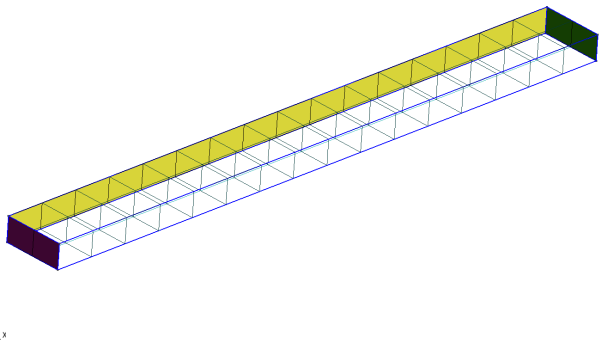


Figure B.12.: Hexahedra

Figure B.13.: Cantilevered beam meshed with structured tetrahedra and hexahedra

Now the following Bash script first calls Gmsh to create the meshes `cantilever-${element}-${c}.msh` where

- `${element}`: tet4, tet10, hex8, hex20, hex27
- `${c}`: 1,2,...,10

It then calls FeenoX with the input above and passes `${element}` and `${c}` as extra arguments, which then are expanded as `$1` and `$2` respectively.

```
#!/bin/bash

rm -f *.dat
for element in tet4 tet10 hex8 hex20 hex27; do
 for c in $(seq 1 10); do

  # create mesh if not alreay cached
  mesh=cantilever-${element}-${c}
  if [ ! -e ${mesh}.msh ]; then
    scale=$(echo "PRINT 1/${c}" | feenox -)
    gmsh -3 -v 0 cantilever-${element}.geo -clscale ${scale} -o ${mesh}.msh
  fi

  # call FeenoX
  feenox cantilever.fee ${element} ${c} | tee -a cantilever-${element}.dat

 done
done
```

After the execution of the script, thanks to the design decision (explained in section B.3.2) that output is 100% defined by the user (in this case with the `PRINT` instruction), one has several files `cantilever-${element}.dat` files. When plotted, these show the shear locking effect of fully-integrated first-order elements as illustrated in figure B.14. The theoretical Euler-Bernoulli result is just a reference as, among other things, it does not take into account the effect of the material's Poisson's ratio. Note that the abscissa shows the number of *nodes*, which are proportional to the number of degrees of freedom (i.e. the size of the problem matrix) and not the number of *elements*, which is irrelevant here and in most problems.

B.2.2.3. Optimization loops

Optimization loops work very much like parametric runs from the FeenoX point of view. The difference is mainly on the calling script that has to implement a certain optimization algorithm such as conjugate gradients, Nelder-Mead, simulated annealing, genetic algorithms, etc. to choose which parameters to pass to FeenoX as command-line argument. The only particularity on FeenoX's side is that since the next argument that the optimization loop will pass might depend on the result of the current step, care has to be taken in order to be able to return back to the calling script whatever results it needs in order to compute the next arguments. This is usually just the scalar being optimized for, but it can also include other results such as derivatives or other relevant data.

To illustrate how to use FeenoX in an optimization loop, let us consider the problem of finding the length ℓ_1 of a tuning fork (figure B.15) such that the fundamental frequency on a free-free oscillation is equal to the base A frequency at 440 Hz.

This extremely simple input file (*rule of simplicity* section C.5) solves the free-free mechanical modal problem (i.e. without any Dirichlet boundary condition) and prints the fundamental frequency:

```
PROBLEM modal 3D MODES 1  # only one mode needed
READ_MESH fork.msh  # in [m]
E = 2.07e11         # in [Pa]
```

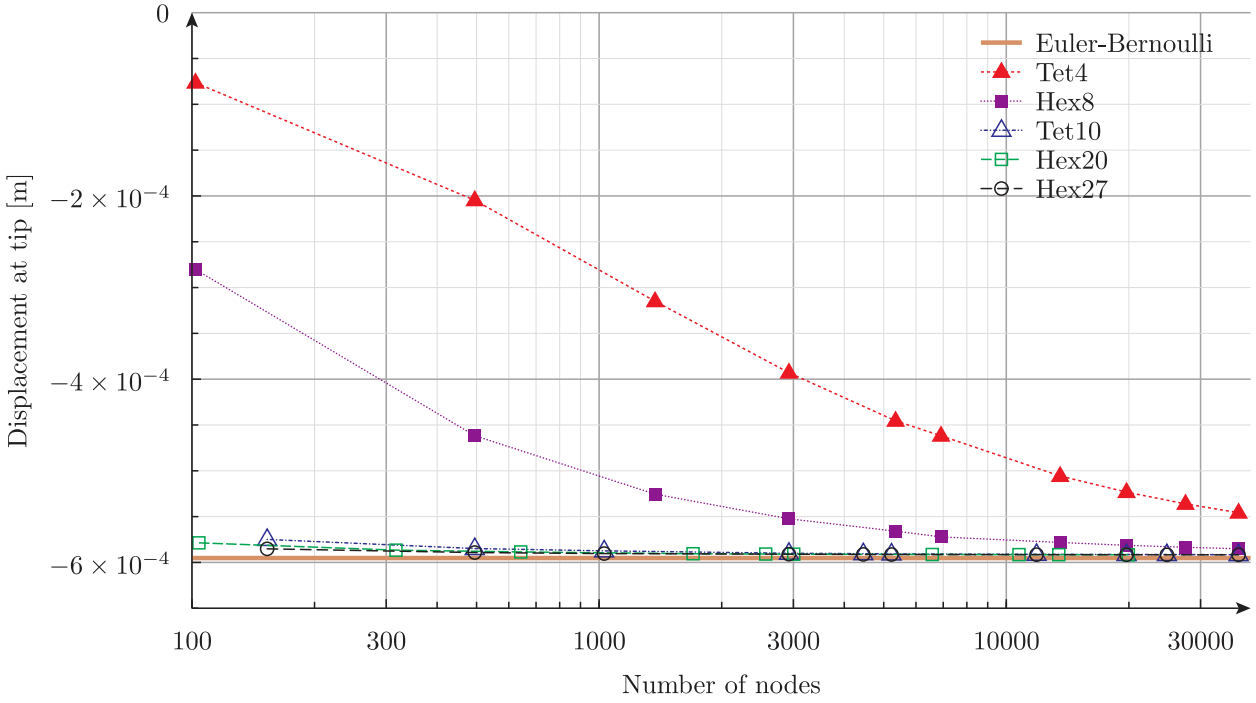


Figure B.14.: Displacement at the free tip of a cantilevered beam vs. number of nodes for different element types

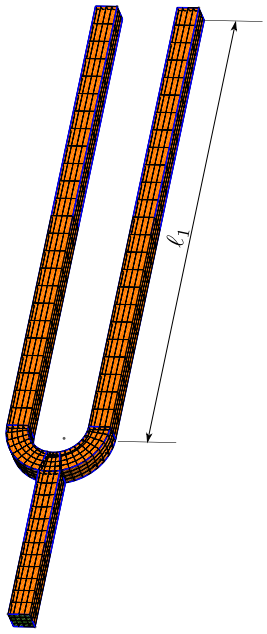


Figure B.15.: What length ℓ_1 is needed so the fork vibrates at 440 Hz?

```
nu = 0.33
rho = 7829          # in [kg/m^2]

# no BCs! It is a free-free vibration problem
SOLVE_PROBLEM

# write back the fundamental frequency to stdout
PRINT f(1)
```

Note that in this particular case, the FeenoX input files does not expand any command-line argument. The trick is that the mesh file `fork.msh` is overwritten in each call of the optimization loop. Since this time the loop is slightly more complex than in the parametric run of the last section, we now use Python. The function `create_mesh()` first creates a CAD model of the fork with geometrical parameters r, w, ℓ_1 and ℓ_2. It then meshes the CAD using n structured hexahedra through the fork's thickness. Both the CAD and the mesh are created using the Gmsh Python API. The detailed steps between `gmsh.initialize()` and `gmsh.finalize()` are not shown here, just the fact that this function overwrites the previous mesh and always writes it into the file called `fork.msh` which is the one that `fork.fee` reads. Hence, there is no need to pass command-liner arguments to FeenoX. The full implementation of the function is available in the examples directory of the FeenoX distribution.

```
import math
import gmsh
import subprocess  # to call FeenoX and read back

def create_mesh(r, w, l1, l2, n):
  gmsh.initialize()
  ...
  gmsh.write("fork.msh")
  gmsh.finalize()
  return len(nodes)

def main():
  target = 440    # target frequency
  eps = 1e-2      # tolerance
  r = 4.2e-3      # geometric parameters
  w = 3e-3
  l1 = 30e-3
  l2 = 60e-3

  for n in range(1,7):   # mesh refinement level
    l1 = 60e-3              # restart l1 & error
    error = 60
    while abs(error) > eps:   # loop
      l1 = l1 - 1e-4*error
      # mesh with Gmsh Python API
      nodes = create_mesh(r, w, l1, l2, n)
      # call FeenoX and read scalar back
      # TODO: FeenoX Python API (like Gmsh)
      result = subprocess.run(['feenox', 'fork.fee'], stdout=subprocess.PIPE)
      freq = float(result.stdout.decode('utf-8'))
      error = target - freq

    print(nodes, l1, freq)
```

Since the computed frequency depends both on the length ℓ_1 and on the mesh refinement level n, there are actually two nested loops: one parametric over $n = 1, 2 \ldots, 7$ and the optimization loop

itself that tries to find ℓ_1 so as to obtain a frequency equal to 440 Hz within 0.01% of error.

```
$ python fork.py > fork.dat
$
```

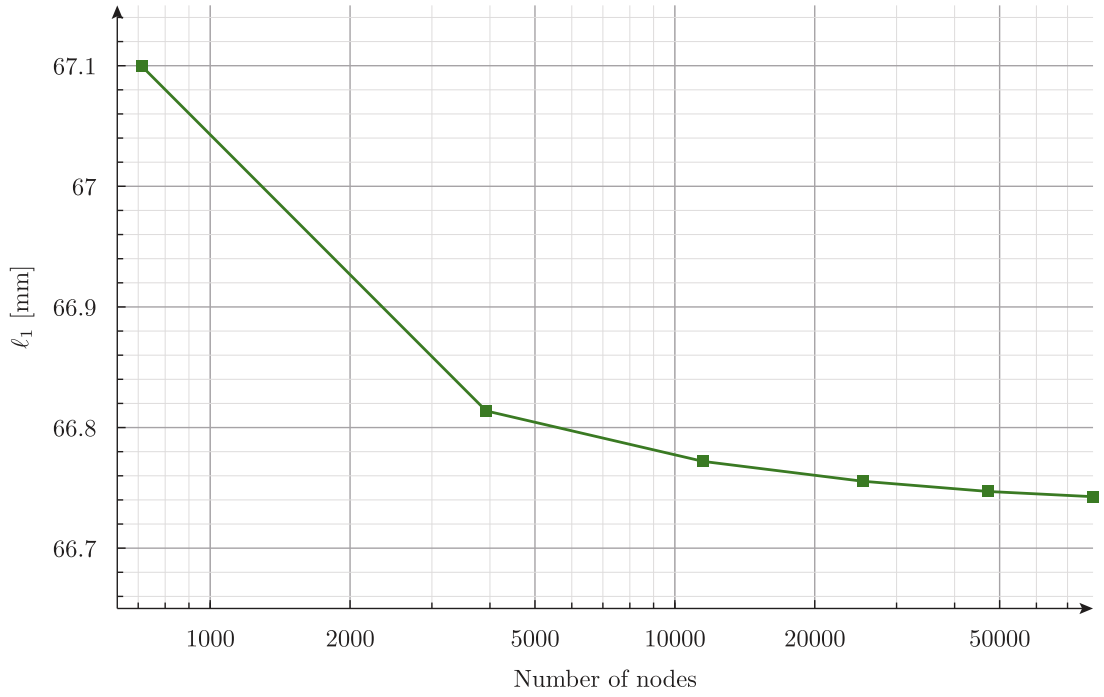


Figure B.16.: Estimated length ℓ_1 needed to get 440 Hz for different mesh refinement levels n

Note that the approach used here is to use Gmsh Python API to build the mesh and then fork the FeenoX executable to solve the fork (no pun intended). There are plans to provide a Python API for FeenoX so the problem can be set up, solved and the results read back directly from the script instead of needing to do a fork+exec, read back the standard output as a string and then convert it to a Python `float`.

Figura B.16 shows the results of the combination of the optimization loop over ℓ_1 and a parametric run over n. The difference for $n = 6$ and $n = 7$ is in the order of one hundredth of millimeter.

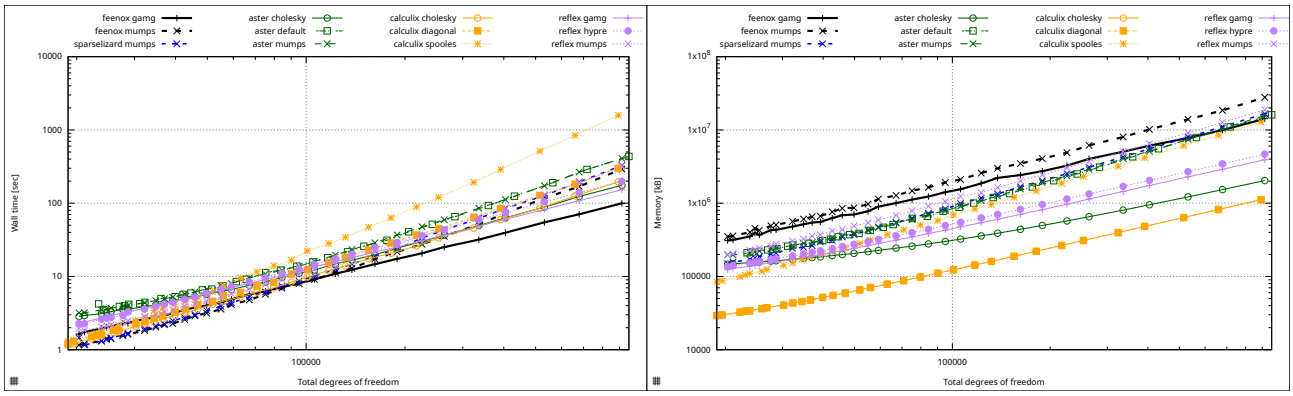
B.2.3. Efficiency

> As required in the previous section, it is mandatory to be able to execute the tool on one or more remote servers. The computational resources needed from this server, i.e. costs measured in
>
> - CPU/GPU time
> - random-access memory
> - long-term storage
> - etc.
>
> needed to solve a problem should be comparable to other similar state-of-the-art cloud-based

> script-friendly finite-element tools.

One of the most widely known quotations in computer science is that one that says "premature optimization is the root of all evil." that is an extremely over-simplified version of Donald E. Knuth's analysis in his The Art of Computer Programming. Bottom line is that the programmer should not not spend too much time trying to optimize code based on hunches but based on profiling measurements. Yet a disciplined programmer can tell when an algorithm will be way too inefficient (say something that scales up like $O(n^2)$) and how small changes can improve performance (say by understanding how caching levels work in order to implement faster nested loops). It is also true that usually an improvement in one aspect leads to a deterioration in another one (e.g. a decrease in CPU time by caching intermediate results in an increase of RAM usage).

Even though FeenoX is still evolving so it could be premature in many cases, it is informative to compare running times and memory consumption when solving the same problem with different cloud-friendly FEA programs. In effect, a serial single-thread single-host comparison of resource usage when solving the NAFEMS LE10 problem introduced above was performed, using both unstructured tetrahedral and structured hexahedral meshes. Figura B.17 shows two figures of the many ones contained in the detailed report. In general, FeenoX using the iterative approach based on PETSc's Geometric-Algebraic Multigrid Preconditioner and a conjugate gradients solver is faster for (relatively) large problems at the expense of a larger memory consumption. The curves that use MUMPS confirm the well-known theoretical result that direct linear solvers are robust but not scalable.



(a) Wall time vs. number of degrees of freedom (b) Memory vs. number of degrees of freedom

Figure B.17.: Resource consumption when solving the NAFEMS LE10 problem in the cloud for tetrahedral meshes.

Regarding storage, FeenoX needs space to store the input file (negligible), the mesh file in `.msh` format (which can be either ASCII or binary) and the optional output files in `.msh` or `.vtk` formats. All of these files can be stored gzip-compressed and un-compressed on demand by exploiting FeenoX's script-friendliness using proper calls to `gzip` before and/or after calling the `feenox` binary.

B.2.4. Scalability

> The tool ought to be able to start solving small problems first to check the inputs and outputs behave as expected and then allow increasing the problem size up in order to achieve to the desired accuracy of the results. As mentioned in section A.2, large problem should be split among different computers to be able to solve them using a finite amount of per-host computational power (RAM and CPU).

When for a fixed problem the mesh is refined over and over, more and more computational resources are needed to solve it (and to obtain more accurate results, of course). Parallelization can help to

a. reduce the wall time needed to solve a problem by using several processors at the same time
b. allow to solve big problems that would not fit into a single computer by splitting them into smaller parts, each of them fitting in a single computer

There are three types of parallelization schemes:

Shared-memory systems (OpenMP) several processing units sharing a single memory address space

Distributed systems (MPI) several computational units, each with their own processing units and memory, inter-connected with high-speed network hardware

Graphical processing units (GPU) used as co-processors to solve numerically-intensive problems

In principle, any of these three schemes can be used to reduce the wall time (a). But only the distributed systems scheme allows to solve arbitrarily big problems (b).

It might seem that the most effective approach to solve a large problem is to use OpenMP (not to be confused with OpenMPI!) among threads running in processors that share the memory address space and to use MPI among processes running in different hosts. But even though this hybrid OpenMP+MPI scheme is possible, there are at least three main drawbacks with respect to a pure MPI approach:

i. the overall performance is not be significantly better
ii. the amount of lines of code that has to be maintained is more than doubled
iii. the number of possible points of synchronization failure increases

In many ways, the pure MPI mode has fewer synchronizations and thus should perform better. Hence, FeenoX uses MPI (mainly through PETSc and SLEPc) to handle large parallel problems.

To illustrate FeenoX's MPI features, let us consider the following input file (which is part of FeenoX's tests suite):

```
PRINTF_ALL "Hello MPI World!"
```

The instruction `PRINTF_ALL` (at the end of the day, it is a verb) asks all the processes to write the `printf`-formatted arguments in the standard output. A prefix is added to each line with the process id and the name of the host. When running FeenoX with this input file through `mpiexec` in an AWS server which has already been properly configured to connect to another one and split the MPI processes, we get:

```
ubuntu@ip-172-31-44-208:~/mpi/hello$ mpiexec --verbose --oversubscribe --hostfile hosts -np 4 ./ ↩
    feenox hello_mpi.fee
[0/4 ip-172-31-44-208] Hello MPI World!
[1/4 ip-172-31-44-208] Hello MPI World!
[2/4 ip-172-31-34-195] Hello MPI World!
[3/4 ip-172-31-34-195] Hello MPI World!
ubuntu@ip-172-31-44-208:~/mpi/hello$
```

That is to say,host `ip-172-31-44-208` spawns two local processes `feenox` and, at the same time, asks host `ip-172-31-34-195` to create two new processes in it. This scheme would allow to solve a problem in parallel where the CPU and RAM loads are split into two different servers.

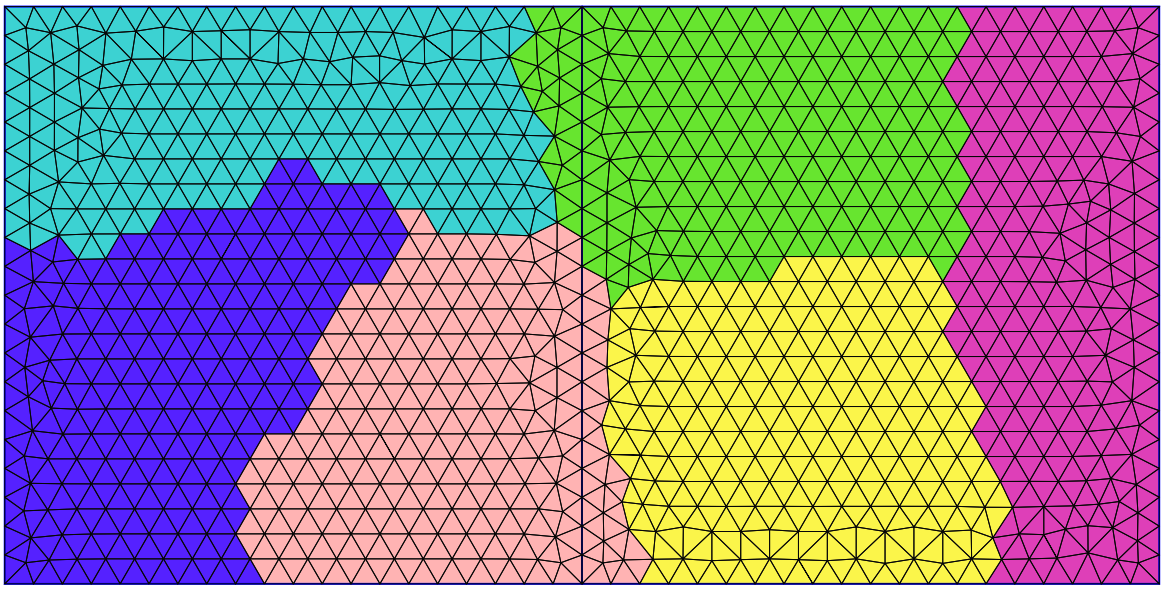


Figure B.18.: Gmsh's tutorial `t21`: two squares decomposed in 6 partitions.

We can used Gmsh's tutorial `t21` that illustrated the concept of domain decomposition (DDM) to show another aspect of how MPI parallelization works in FeenoX. In effect, let us consider the mesh from figure B.18 that consists of two non-dimensional squares of size 1×1 and let us say we want to compute the integral of the constant 1 over the surface to obtain the numerical result 2.

```
READ_MESH t21.msh
INTEGRATE 1 RESULT two
PRINTF_ALL "%g" two
```

In this case, the instruction `INTEGRATE` is executed in parallel where each process computes the local contribution and, before moving into the next instruction (`PRINTF_ALL`), all processes synchronize and sum up all these contributions (i.e. they perform a sum reduction) and all the processes obtain the global result in the variable `two`:

```
$ mpiexec -n 2 feenox t21.fee
[0/2 tom] 2
[1/2 tom] 2
$ mpiexec -n 4 feenox t21.fee
[0/4 tom] 2
```

```
[1/4 tom] 2
[2/4 tom] 2
[3/4 tom] 2
$ mpiexec -n 6 feenox t21.fee
[0/6 tom] 2
[1/6 tom] 2
[2/6 tom] 2
[3/6 tom] 2
[4/6 tom] 2
[5/6 tom] 2
$
```

To illustrate what is happening under the hood, let us temporarily modify the FeenoX source code so that each process shows the local contribution:

```
$ mpiexec -n 2 feenox t21.fee
[process 0] my local integral is 0.996699
[process 1] my local integral is 1.0033
[0/2 tom] 2
[1/2 tom] 2
$ mpiexec -n 3 feenox t21.fee
[process 0] my local integral is 0.658438
[process 1] my local integral is 0.672813
[process 2] my local integral is 0.668749
[0/3 tom] 2
[1/3 tom] 2
[2/3 tom] 2
$ mpiexec -n 4 feenox t21.fee
[process 0] my local integral is 0.505285
[process 1] my local integral is 0.496811
[process 2] my local integral is 0.500788
[process 3] my local integral is 0.497116
[0/4 tom] 2
[1/4 tom] 2
[2/4 tom] 2
[3/4 tom] 2
$ mpiexec -n 5 feenox t21.fee
[process 0] my local integral is 0.403677
[process 1] my local integral is 0.401883
[process 2] my local integral is 0.399116
[process 3] my local integral is 0.400042
[process 4] my local integral is 0.395281
[0/5 tom] 2
[1/5 tom] 2
[2/5 tom] 2
[3/5 tom] 2
[4/5 tom] 2
$ mpiexec -n 6 feenox t21.fee
[process 0] my local integral is 0.327539
[process 1] my local integral is 0.330899
[process 2] my local integral is 0.338261
[process 3] my local integral is 0.334552
[process 4] my local integral is 0.332716
[process 5] my local integral is 0.336033
[0/6 tom] 2
```

```
[1/6 tom] 2
[2/6 tom] 2
[3/6 tom] 2
[4/6 tom] 2
[5/6 tom] 2
$
```

Note that in the cases with 4 and 5 processes, the number of partitions P is not a multiple of the number of processes N. Anyway, FeenoX is able to distribute the load is able to distribute the load among the N processes, even though the efficiency is slightly less than in the other cases. :::

When solving PDEs, FeenoX builds the local matrices and vectors and then asks PETSc to assemble the global objects by sending non-local information as MPI messages. This way, all processes have contiguous rows as local data and the system of equations can be solved in parallel using the distributed system paradigm.

We can show that both

a. the wall time, and
b. the per-process memory

decrease when running a fixed-sized problem with MPI in parallel using the IAEA 3D PWR benchmark:

```
PROBLEM neutron_diffusion 3D GROUPS 2

DEFAULT_ARGUMENT_VALUE 1 quarter
READ_MESH iaea-3dpwr-$1.msh

MATERIAL fuel1     D1=1.5 D2=0.4 Sigma_s1.2=0.02 Sigma_a1=0.01 Sigma_a2=0.08  nuSigma_f2=0.135
MATERIAL fuel2     D1=1.5 D2=0.4 Sigma_s1.2=0.02 Sigma_a1=0.01 Sigma_a2=0.085 nuSigma_f2=0.135
MATERIAL fuel2rod  D1=1.5 D2=0.4 Sigma_s1.2=0.02 Sigma_a1=0.01 Sigma_a2=0.13  nuSigma_f2=0.135
MATERIAL reflector D1=2.0 D2=0.3 Sigma_s1.2=0.04 Sigma_a1=0    Sigma_a2=0.01  nuSigma_f2=0
MATERIAL reflrod   D1=2.0 D2=0.3 Sigma_s1.2=0.04 Sigma_a1=0    Sigma_a2=0.055 nuSigma_f2=0

BC vacuum   vacuum=0.4692
BC mirror   mirror

SOLVE_PROBLEM
WRITE_RESULTS FORMAT vtk

PRINT  "geometry = $1"
PRINTF "    keff = %.5f"     keff
PRINTF "   nodes = %g"       nodes
PRINTF "    DOFs = %g"       total_dofs
PRINTF "  memory = %.1f Gb (local) %.1f Gb (global)" mpi_memory_local() mpi_memory_global()
PRINTF "    wall = %.1f sec" wall_time()
```

```
$ mpiexec -n 1 feenox iaea-3dpwr.fee quarter
geometry = quarter
    keff = 1.02918
   nodes = 70779
    DOFs = 141558
[0/1 tux]   memory = 2.3 Gb (local) 2.3 Gb (global)
    wall = 26.2 sec
$ mpiexec -n 2 feenox iaea-3dpwr.fee quarter
```

```
geometry = quarter
    keff = 1.02918
   nodes = 70779
    DOFs = 141558
[0/2 tux]   memory = 1.5 Gb (local) 3.0 Gb (global)
[1/2 tux]   memory = 1.5 Gb (local) 3.0 Gb (global)
    wall = 17.0 sec
$ mpiexec -n 4 feenox iaea-3dpwr.fee quarter
geometry = quarter
    keff = 1.02918
   nodes = 70779
    DOFs = 141558
[0/4 tux]   memory = 1.0 Gb (local) 3.9 Gb (global)
[1/4 tux]   memory = 0.9 Gb (local) 3.9 Gb (global)
[2/4 tux]   memory = 1.1 Gb (local) 3.9 Gb (global)
[3/4 tux]   memory = 0.9 Gb (local) 3.9 Gb (global)
    wall = 13.0 sec
$
```

B.2.5. Flexibility

> The tool should be able to handle engineering problems involving different materials with potential spatial and time-dependent properties, such as temperature-dependent thermal expansion coefficients and/or non-constant densities. Boundary conditions must be allowed to depend on both space and time as well, like non-uniform pressure loads and/or transient heat fluxes.

The third-system effect mentioned in section B.2 involves more than ten years of experience in the nuclear industry,[4] where complex dependencies of multiple material properties over space through intermediate distributions (temperature, neutronic poisons, etc.) and time (control rod positions, fuel burn-up, etc.) are mandatory. One of the cornerstone design decisions in FeenoX is that **everything is an expression** (section B.3.1.5). Here, "everything" means any location in the input file where a numerical value is expected. The most common use case is in the `PRINT` keyword. For example, the Sophomore's dream (in contrast to Freshman's dream) identity

$$\int_0^1 x^{-x}\,dx = \sum_{n=1}^{\infty} n^{-n}$$

can be illustrated like this:

```
VAR x
PRINT %.7f integral(x^(-x),x,0,1)
VAR n
PRINT %.7f sum(n^(-n),n,1,1000)
```

[4] This experience also shaped many of the features that FeenoX has and most of the features is does deliberately not have.

```
$ feenox sophomore.fee
1.2912861
1.2912860
$
```

Of course most engineering problems will not need explicit integrals—although a few of them do—but some might need summation loops, so it is handy to have these functionals available inside the FEA tool. This might seem to go against the "keep it simple" and "do one thing good" Unix principle, but definitely follows Alan Kay's idea that "simple things should be simple, complex things should be possible" (further discussion in section B.3.1.4).

Flexibility in defining non-trivial material properties is illustrated with the following example, where two squares made of different dimensionless materials are juxtaposed in thermal contact (glued?) and subject to different boundary conditions at each of the four sides (figure B.19).

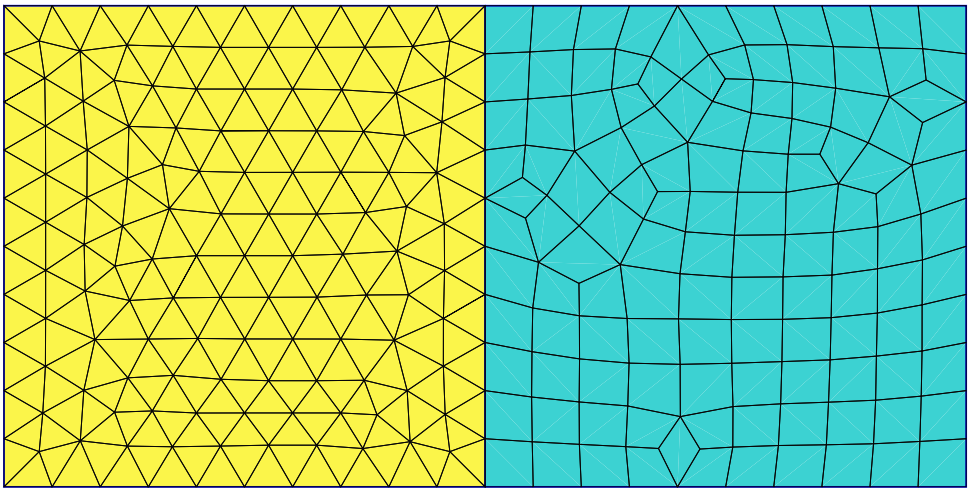


Figure B.19.: Two non-dimensional 1×1 squares each in thermal contact made of different materials.

The yellow square is made of a certain material with a conductivity that depends algebraically (and fictitiously) the temperature like

$$k_{\text{yellow}}(x,y) = \frac{1}{2} + T(x,y)$$

The cyan square has a space-dependent temperature given by a table of scattered data as a function of the spatial coordinates x and y (origin is left bottom corner of the yellow square) without any particular structure on the definition points:

x	y	$k_{\text{cyan}}(x,y)$
1	0	1.0
1	1	1.5
2	0	1.3
2	1	1.8
1.5	0.5	1.7

The cyan square generates a temperature-dependent power density (per unit area) given by

$$q''_{\text{cyan}}(x, y) = 0.2 \cdot T(x, y)$$

The yellow one does not generate any power so $q''_{\text{yellow}} = 0$. Boundary conditions are

$$\begin{cases} T(x,y) = y & \text{at the left edge } y = 0 \\ T(x,y) = 1 - \cos\left(\frac{1}{2}\pi \cdot x\right) & \text{at the bottom edge } x = 0 \\ q'(x,y) = 2 - y & \text{at the right edge } x = 2 \\ q'(x,y) = 1 & \text{at the top edge } y = 1 \end{cases}$$

The input file illustrate how flexible FeenoX is and, again, how the problem definition in a format that the computer can understand resembles the humanly-written formulation of the original engineering problem:

```
PROBLEM thermal 2d            # heat conduction in two dimensions
READ_MESH two-squares.msh

k_yellow(x,y) = 1/2+T(x,y)    # thermal conductivity
FUNCTION k_cyan(x,y) INTERPOLATION shepard DATA {
    1   0    1.0
    1   1    1.5
    2   0    1.3
    2   1    1.8
    1.5 0.5  1.7 }

q_cyan(x,y) = 1-0.2*T(x,y)    # dissipated power density
q_yellow(x,y) = 0

BC left   T=y                 # temperature (dirichlet) bc
BC bottom T=1-cos(pi/2*x)
BC right  q=2-y               # heat flux (neumann) bc
BC top    q=1

SOLVE_PROBLEM
WRITE_MESH two-squares-results.msh  T #CELLS k
```

Note that FeenoX is flexible enough to...

1. handle mixed meshes (the yellow square is meshed with triangles and the other one with quadrangles)
2. use point-wise defined properties even though there is not underlying structure nor topology for the points where the data is defined (FeenoX could have read data from a `.msh` or `.vtk` file respecting the underlying topology)
3. understand that the problem is non-linear so as to use PETSc's SNES framework automatically (the conductivity and power source depend on the temperature).

In the very same sense that variables `x`, `y` and `z` appearing in the input refer to the spatial coordinates x, y and z respectively, the special variable `t` refers to the time t. The requirement of allowing time-dependent boundary conditions can be illustrated by solving the NAFEMS T3 one-dimensional

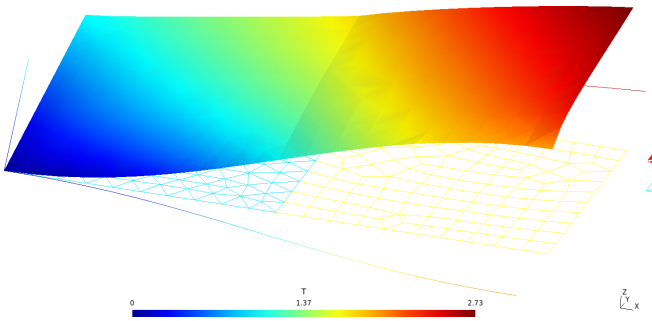


Figure B.20.: Temperature defined at nodes

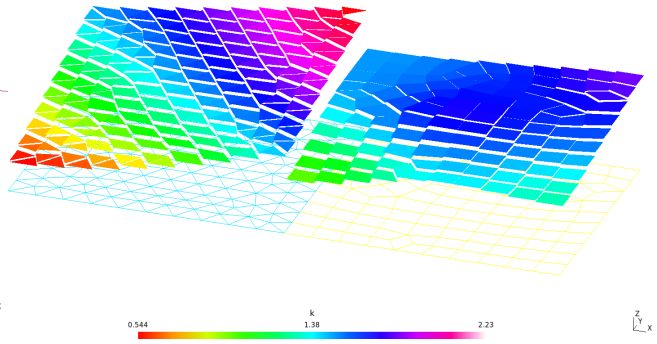


Figure B.21.: Conductivity defined at cells

Figure B.22.: Temperature (main result) and conductivity for the two-squares thermal problem.

transient heat transfer benchmark. It consists of a slab of 0.1 meters long subject to a fixed homogeneous temperature on one side, i.e.

$$T(x = 0) = 0\ ^\circ\text{C}$$

and to a transient temperature

$$T(x = 0.1\ \text{m}, t) = 100\ ^\circ\text{C} \cdot \sin\left(\frac{\pi \cdot t}{40\ \text{s}}\right)$$

at the other side. There is zero internal heat generation, at $t = 0$ all temperature is equal to 0°C (sic) and conductivity, specific heat and density are constant and uniform. The problem asks for the temperature at location $x = 0.08$ m at time $t = 32$ s. The reference result is $T(0.08\ \text{m}, 32\ \text{s}) = 36.60\ ^\circ\text{C}$.

```
PROBLEM thermal DIM 1 # NAFEMS-T3 benchmark: 1d transient heat conduction
READ_MESH slab-0.1m.msh

end_time = 32       # transient up to 32 seconds
T_0(x) = 0          # initial condition "all temperature is equal to 0°C"

# prescribed temperatures as boundary conditions
BC left  T=0
BC right T=100*sin(pi*t/40)

# uniform and constant properties
k = 35.0            # conductivity [W/(m K)]
cp = 440.5          # heat capacity [J/(kg K)]
rho = 7200          # density [kg/m^3]

SOLVE_PROBLEM

# print detailed evolution into an ASCII file
PRINT FILE nafems-t3.dat %.3f t dt %.2f T(0.05) T(0.08) T(0.1)

# print the asked result into the standard output
IF done
 PRINT "T(0.08m,32s) = " T(0.08) "ºC"
ENDIF
```

```
$ gmsh -1 slab-0.1m.geo
[...]
Info    : Done meshing 1D (Wall 0.000213023s, CPU 0.000836s)
Info    : 61 nodes 62 elements
Info    : Writing 'slab-0.1m.msh'...
Info    : Done writing 'slab-0.1m.msh'
Info    : Stopped on Sun Dec 12 19:41:18 2021 (From start: Wall 0.00293443s, CPU 0.02605s)
$ feenox nafems-t3.fee
T(0.08m,32s) =  36.5996 ºC
$ pyxplot nafems-t3.ppl
$
```

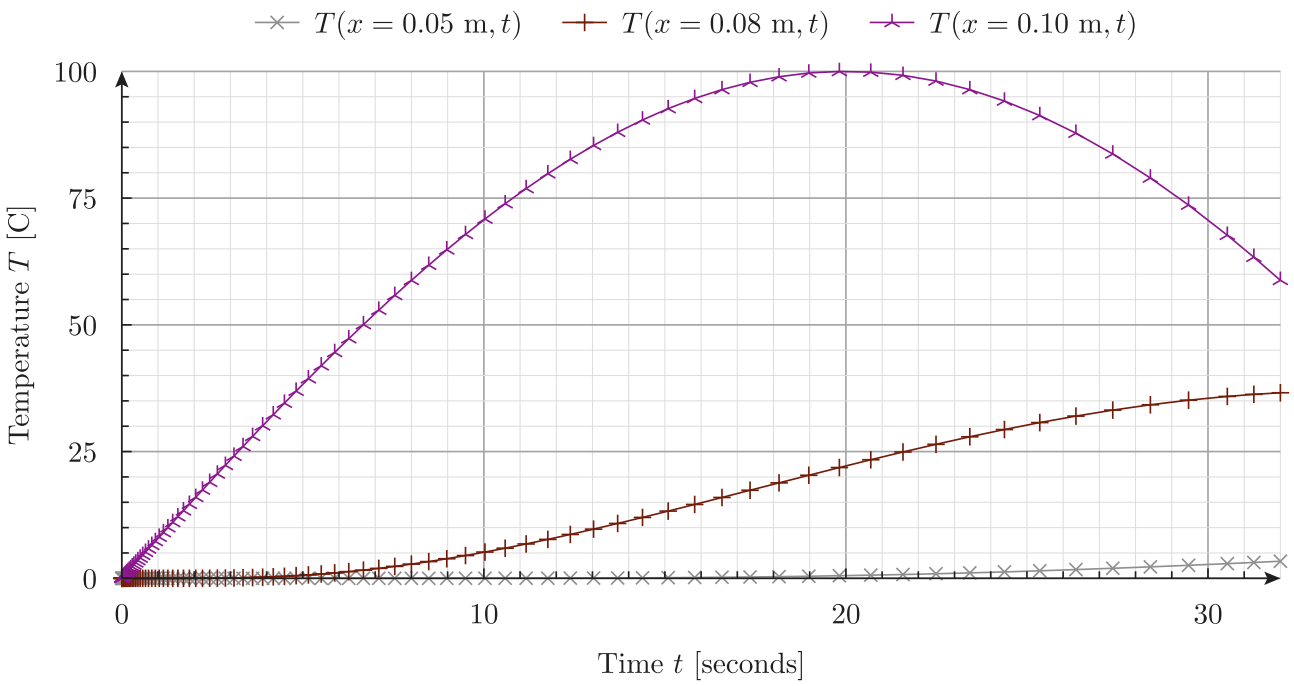


Figure B.23.: Temperature vs. time at three axial locations for the NAFEMS T3 benchmark

Besides "everything is an expression," FeenoX follows another cornerstone rule: **simple problems ought to have simple inputs**, akin to Unix' *rule of simplicity*—that addresses the first half of Alan Kay's quote above. This rule is further discussed in section B.3.1.

B.2.6. Extensibility

> It should be possible to add other problem types casted as PDEs (such as the Schröedinger equation) to the tool using a reasonable amount of time by one or more skilled programmers. The tool should also allow new models (such as non-linear stress-strain constitutive relationships) to be added as well.

When solving partial differential equations numerically, there are some steps that are independent of the type of PDE. For example,

1. read the mesh

2. evaluate the coefficients (i.e. material properties)
3. solve the discretized systems of algebraic equations
4. write the results

Even though FeenoX is written in C, it makes extensive use of function pointers to mimic C++'s virtual methods. This way, depending on the problem type given with the `PROBLEM` keyword, particular PDE-specific routines are called to

1. initialize and set up solver options (steady-state/transient, linear/non-linear, regular/eigen-problem, etc.)
2. parse boundary conditions given in the `BC` keyword
3. build elemental contributions for
    a. volumetric stiffness and/or mass matrices
    b. natural boundary conditions
4. compute secondary fields (heat fluxes, strains and stresses, etc.) out of the gradients of the primary fields
5. compute per-problem key performance indicators (min/max temperature, displacement, stress, etc.)
6. write particular post-processing outputs

Indeed, each of the supported problems, namely

- `laplace`
- `thermal`
- `mechanical`
- `modal`
- `neutron_diffusion`
- `neutron_sn`

is a separate directory under `src/pdes` that implements these "virtual" methods (recall that they are function pointers) that are resolved at runtime when parsing the main input file.

FeenoX was designed with separated common "mathematical" routines from the particular "physical" ones in such a way that any of these directories can be removed and the code would still compile. The `autogen.sh` is in charge of

1. parsing the source tree
2. detect which are the available PDEs
3. create appropriate snippets of code so the common mathematical framework can resolve the function pointers for the entry points
4. build the `Makefile.am` templates used by the `configure` script

For example, if we removed the directory `src/pdes/thermal` from a temporary clone of the main Git repository then the whole bootstrapping, configuration and compilation procedure would produce a `feenox` executable without the ability to solve thermal problems:

```
~$ cd tmp/
~/tmp$ git clone https://github.com/seamplex/feenox
Cloning into 'feenox'...
remote: Enumerating objects: 6908, done.
remote: Counting objects: 100% (4399/4399), done.
remote: Compressing objects: 100% (3208/3208), done.
remote: Total 6908 (delta 3085), reused 2403 (delta 1126), pack-reused 2509
Receiving objects: 100% (6908/6908), 10.94 MiB | 6.14 MiB/s, done.
Resolving deltas: 100% (4904/4904), done.
~/tmp$ cd feenox
~/tmp/feenox$ rm -rf src/pdes/thermal/
~/tmp/feenox$ ./autogen.sh
creating Makefile.am... ok
creating src/Makefile.am... ok
calling autoreconf...
configure.ac:18: installing './compile'
configure.ac:15: installing './config.guess'
configure.ac:15: installing './config.sub'
configure.ac:17: installing './install-sh'
configure.ac:17: installing './missing'
parallel-tests: installing './test-driver'
src/Makefile.am: installing './depcomp'
done
~/tmp/feenox$ ./configure.sh
[...]
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating Makefile
config.status: executing depfiles commands
~/tmp/feenox$ make
[...]
make[1]: Leaving directory '/home/[redacted]/tmp/feenox'
~/tmp/feenox$
```

Now if we wanted to run the thermal problem with the two juxtaposed squares from section B.2.5 above, the "temporary" FeenoX would complain. But it would still be able solve the NAFEMS LE10 problem problem right away:

```
~/tmp/feenox$ cd
~/tmp/feenox/doc$ ../feenox two-squares.fee
error: two-squares.fee: 1: unknown problem type 'thermal'
~/tmp/feenox/doc$ cd ../examples
~/tmp/feenox/examples$ ../feenox nafems-le10.fee
sigma_y @ D =   -5.38367        MPa
~/tmp/feenox/examples$
```

The list of available PDEs that a certain FeenoX binary has can be found by using the `--pdes` option. They are sorted alphabetically, one type per line:

```
~/tmp/feenox/examples$ feenox --pdes
laplace
mechanical
modal
```

```
neutron_diffusion
~/tmp/feenox/examples$
```

Besides removals, additions—which are also handled by `autogen.sh` as describe above—are far more interesting to discuss. Additional elliptic problems can be added by using the `laplace` directory as a template while using the other directories as examples about how to add further features (e.g. a Robin-type boundary condition in `thermal` and a vector-valued unknown in `mechanical`). More information can be found in the FeenoX programming & contributing section.

As already discussed in section B.1, FeenoX is free-as-in-freedom software licensed under the terms of the GNU General Public License version 3 or, at the user convenience, any later version. In the particular case of additions to the code base, this fact has two implications.

i. Every person in the world is *free* to modify FeenoX to suit their needs, including adding a new problem type either by

   a. using one of the existing ones as a template, or
   b. creating a new directory from scratch

   without asking anybody for any kind of permission. In case this person does not how to program, he or she has the *freedom* to hire somebody else to do it. It is this the sense of the word "free" in the compound phrase "free software:" freedom to do what they think fit (except to make it non-free, see next bullet).

ii. People adding code own the copyright of the additional code. Yet, if they want to distribute the modified version they have to do it also under the terms of the GPLv3+ and under a name that does not induce the users to think the modified version is the original FeenoX distribution.[5] That is to say, free software ought to remain free—a.k.a. as copyleft.

Regarding additional material models, the virtual methods that compute the elemental contributions to the stiffness matrix also use function pointers to different material models (linear isotropic elastic, orthotropic elastic, etc.) and behaviors (isotropic thermal expansion, orthotropic thermal expansion, etc.) that are resolved at run time. Following the same principle, new models can be added by adding new routines and resolving them depending on the user's input.

B.2.7. Interoperability

> A mean of exchanging data with other computational tools complying to requirements similar to the ones outlined in this document. This includes pre and post-processors but also other computational programs so that coupled calculations can be eventually performed by efficiently exchanging information between calculation codes.

Sección B.1.2 already introduced the ideas about interoperability behind the Unix philosophy which make up for most the the FeenoX design basis. Essentially, they sum up to "do only one thing but do it well." Since FeenoX is filter (or a transfer-function), interoperability is a must. So far, this SDS has already shown examples of exchanging information with:

[5] Even better, these authors should ask to merge their contributions into FeenoX's main code base.

- Kate (with syntax highlighting): figure B.3
- Gmsh (both as a mesher and a post-processor): [figure B.8, figura B.9, figura B.13, figura B.15, figura B.19, figura B.22]
- Paraview: figure B.4
- Gnuplot: [figure B.6, figura B.17]
- Pyxplot: [figure B.14, figura B.16, figura B.23]

To illustrate this approach, consider the following input file that solves Laplace's equation $\nabla^2\phi = 0$ on a square with some space-dependent boundary conditions. Either Gmsh or Paraview can be used to post-process the results:

$$\begin{cases} \phi(x,y) = +y & \text{for } x = -1 \text{ (left)} \\ \phi(x,y) = -y & \text{for } x = +1 \text{ (right)} \\ \nabla\phi \cdot \hat{\mathbf{n}} = \sin\left(\frac{\pi}{2} \cdot x\right) & \text{for } y = -1 \text{ (bottom)} \\ \nabla\phi \cdot \hat{\mathbf{n}} = 0 & \text{for } y = +1 \text{ (top)} \end{cases}$$

```
PROBLEM laplace 2d
READ_MESH square-centered.msh # [-1:+1]x[-1:+1]

# boundary conditions
BC left    phi=+y
BC right   phi=-y
BC bottom  dphidn=sin(pi/2*x)
BC top     dphidn=0

SOLVE_PROBLEM

# same output in .msh and in .vtk formats
WRITE_MESH laplace-square.msh phi VECTOR dphidx dphidy 0
WRITE_MESH laplace-square.vtk phi VECTOR dphidx dphidy 0
```

A great deal of FeenoX interoperability capabilities comes from another design decision: **output is 100% controlled by the user** (further discussed in section B.3.2), a.k.a. "no `PRINT`, no OUTPUT" whose corollary is the Unix *rule of silence* (section C.11). The following input file computes the natural frequencies of oscillation of a cantilevered wire both using the Euler-Bernoulli theory and finite elements. It writes a Gihub-formatted markdown table into the standard output which is then piped to Pandoc and then converted to HTML:

```
# compute the first five natural modes of a cantilever wire
# see https://www.seamplex.com/docs/alambre.pdf (in Spanish)
# (note that there is a systematic error of a factor of two in the measured values)
# see ↩
    https://www.seamplex.com/feenox/examples/modal.html#five-natural-modes-of-a-cantilevered-wire
# for a slighly more complex example

# wire geometry
l = 0.5*303e-3   # [ m ] cantilever length
d = 1.948e-3     # [ m ] diameter

# material properties for copper
mass = 0.5*8.02e-3       # [ kg ] total mass (half the measured because of the experimental ↩
    disposition)
volume = pi*(0.5*d)^2*l
```

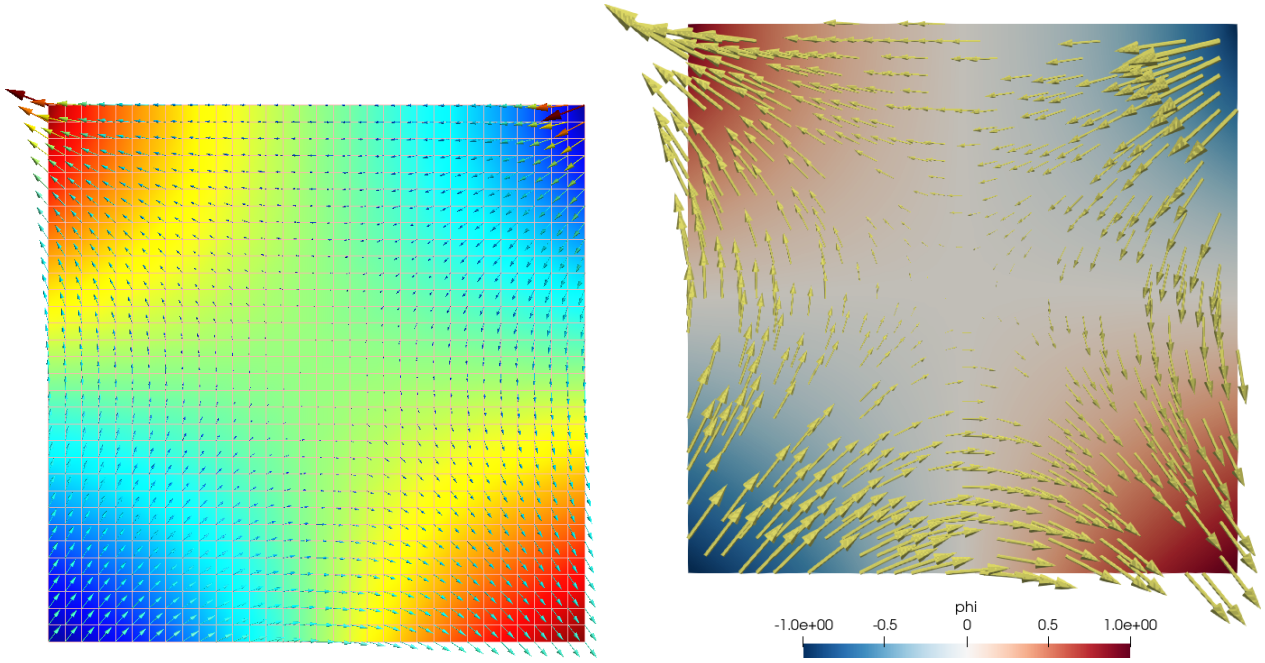


Figure B.24.: Laplace's equation solved with FeenoX

```
rho = mass/volume          # [ kg / m^3 ] density = mass (measured) / volume
E = 2*66.2e9               # [ Pa ] Young modulus (twice because the factor-two error)
nu = 0                     # 'Poissons ratio (does not appear in Euler-Bernoulli)

# compute analytical solution
# first compute the first five roots ok cosh(kl)*cos(kl)+1
VECTOR kl[5]
kl[i] = root(cosh(t)*cos(t)+1, t, 3*i-2,3*i+1)

# then compute the frequencies according to Euler-Bernoulli
# note that we need to use SI inside the square root
A = pi * (d/2)^2
I = pi/4 * (d/2)^4

VECTOR f_euler[5]
f_euler[i] = 1/(2*pi) * kl(i)^2 * sqrt((E * I)/(rho * A * l^4))

# now compute the modes numerically with FEM
# note that each mode is duplicated as it is degenerated
PROBLEM modal 3D MODES 10
READ_MESH wire-hex.msh
BC fixed fixed
SOLVE_PROBLEM


# write a github-formatted markdown table comparing the frequencies
PRINT "  \$n\$ |   FEM  | Euler | Relative difference [%]"
PRINT ":----:+:------:+:-----:+:----------------------:"
PRINT_VECTOR SEP " | "  %g i  %.4g f(2*i-1) f_euler   %.2f 100*(f_euler(i)-f(2*i-1))/f_euler(i)
PRINT
PRINT ": Comparison of analytical and numerical frequencies, in Hz"
```

```
$ gmsh -3 wire-hex.geo
[...]
```

```
$ feenox wire.fee | pandoc
<table>
<caption>Comparison of analytical and numerical frequencies, in Hz</caption>
<thead>
<tr class="header">
<th style="text-align: center;"><span class="math inline"><em>n</em></span></th>
<th style="text-align: center;">FEM</th>
<th style="text-align: center;">Euler</th>
<th style="text-align: center;">Relative difference [%]</th>
</tr>
</thead>
<tbody>
<tr class="odd">
<td style="text-align: center;">1</td>
<td style="text-align: center;">45.84</td>
<td style="text-align: center;">45.84</td>
<td style="text-align: center;">0.02</td>
</tr>
<tr class="even">
<td style="text-align: center;">2</td>
<td style="text-align: center;">287.1</td>
<td style="text-align: center;">287.3</td>
<td style="text-align: center;">0.06</td>
</tr>
<tr class="odd">
<td style="text-align: center;">3</td>
<td style="text-align: center;">803.4</td>
<td style="text-align: center;">804.5</td>
<td style="text-align: center;">0.13</td>
</tr>
<tr class="even">
<td style="text-align: center;">4</td>
<td style="text-align: center;">1573</td>
<td style="text-align: center;">1576</td>
<td style="text-align: center;">0.24</td>
</tr>
<tr class="odd">
<td style="text-align: center;">5</td>
<td style="text-align: center;">2596</td>
<td style="text-align: center;">2606</td>
<td style="text-align: center;">0.38</td>
</tr>
</tbody>
</table>
$
```

Of course these kind of FeenoX-generated tables can be inserted verbatim into Markdown documents (just like this one) and rendered as tabla B.4.

n	FEM	Euler	Relative difference [%]
1	45.84	45.84	0.02
2	287.1	287.3	0.06
3	803.4	804.5	0.13

n	FEM	Euler	Relative difference [%]
4	1573	1576	0.24
5	2596	2606	0.38

Tabla B.4.: Comparison of analytical and numerical frequencies, in Hz

It should be noted that all of the programs and tools mentioned to be interoperable with FeenoX are free and open source software. This is not a requirement from the SRS, but is indeed a nice-to-have feature.

B.3. Interfaces

> The tool should be able to allow remote execution without any user intervention after the tool is launched. To achieve this goal it is required that the problem should be completely defined in one or more input files and the output should be complete and useful after the tool finishes its execution, as already required. The tool should be able to report the status of the execution (i.e. progress, errors, etc.) and to make this information available to the user or process that launched the execution, possibly from a remote location.

FeenoX is provided as a console-only executable (recall it is a program, not a library) which can be run remotely through the mechanisms discussed in section B.2.2 without any requirement such as graphical servers or special input devices. As already explained, when executed without any arguments, FeenoX writes a brief message with the version (further discussed in section B.4.1) and the basic usage on the standard output and return to the calling shell with a return errorlevel zero:

```
$ feenox
FeenoX v0.3.292-gc932cb5
a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool

usage: feenox [options] inputfile [replacement arguments] [petsc options]

  -h, --help         display options and detailed explanations of commmand-line usage
  -v, --version      display brief version information and exit
  -V, --versions     display detailed version information
  --pdes             list the types of PROBLEMs that FeenoX can solve, one per line
  --elements_info    output a document with information about the supported element types
  --linear           force FeenoX to solve the PDE problem as linear
  --non-linear       force FeenoX to solve the PDE problem as non-linear

Run with --help for further explanations.
$ echo $?
0
$
```

The `--version` option follows the GNU Coding Standards guidelines:

Pair A	Pair B	Applied Cycles A	Applied Cycles B	M+B STRESS (psi)	K_e	Total Stress (psi)	S_{alt} (psi)	N_n	n_n	U_n	Max. Metal Temp. (°F)	DO (ppm)
694	447	5	20	125542.9	2.580	144164.4	220490.4	140.005	5	0.0357	566.6	0.150
699	447	50	15	121622.8	2.405	139047.0	198300.6	178.958	15	0.0838	566.6	0.550
699	1021	35	20	104691.5	1.653	126037.5	124507.0	582.468	20	0.0343	600.4	0.550
699	899	15	50	89695.4	1.000	102302.8	57864.5	6339.47	15	0.0024	336.1	0.550
695	899	5	35	84993.9	1.000	98798.6	55882.4	7027.83	5	0.0007	336.1	0.550
185	899	20	30	68222.2	1.000	76465.1	43250.2	15549.1	20	0.0013	336.1	0.550
1432	899	20	10	66665.7	1.000	83098.8	47002.3	11892.7	10	0.0008	336.1	0.550
1432	1653	10	100	49437.0	1.000	61950.9	33687.5	35734.8	10	0.0003	103.0	0.522
1296	1653	20	90	32478.6	1.000	38719.1	22025.4	154852	20	0.0001	366.2	0.522
1136	1653	20	70	27045.6	1.000	33751.1	19388.7	258499	20	0.0001	417.7	0.522
2215	1653	100	50	25255.9	1.000	25668.1	15147.6	1.15E+06	50	0.0000	547.0	0.522
2215	1213	50	20	22343.7	1.000	25298.3	14929.4	1.30E+06	20	0.0000	547.0	0.050
2215	1562	30	20	22047.7	1.000	24970.1	14735.7	1.46E+06	20	0.0000	547.0	0.050
2215	1	10	20	11956.0	1.000	12255.6	7232.5	1.00E+11	10	0.0000	547.0	0.150
1347	1	20	10	3786.5	1.000	4173.0	2412.1	1.00E+11	10	0.0000	450.0	0.150
1347	1595	10	20	3408.0	1.000	3430.2	1963.3	1.00E+11	10	0.0000	398.7	0.050
960	1595	20	10	241.8	1.000	259.9	146.0	1.00E+11	10	0.0000	299.5	0.050
960	960	5	5	0.0	1.000	0.0	0.0	1.00E+11	10	0.0000	299.5	0.050
										TOTAL CUF =	**0.1596**	

(a) A multi-billion-dollar agency using the Windows philosophy (presumably mouse-based copy and pasted into Word)

j	A_j	B_j	$n(A_j)$	$n(B_j)$	MB'_j [ksi]	$k_{e,j}$	S'_j [ksi]	$S_{\mathrm{alt},j}$ [ksi]	N_j	n_j	U_j	$T_{\max,j}$ [°F]
1	447	694	20	5	125.5	2.580	144.2	220.400	1.40×10^{2}	5	3.57×10^{-2}	566.6
2	447	699	15	50	121.6	2.405	139	198.300	1.79×10^{2}	15	8.38×10^{-2}	566.6
3	699	1020	35	20	104.7	1.653	126.5	124.900	5.77×10^{2}	20	3.47×10^{-2}	599.2
4	699	899	15	50	89.7	1.000	102.3	62.640	5.02×10^{3}	15	2.99×10^{-3}	336.1
5	695	899	5	35	84.99	1.000	98.8	59.750	5.77×10^{3}	5	8.67×10^{-4}	336.1
6	899	1432	30	20	66.67	1.000	83.1	50.360	9.56×10^{3}	20	2.09×10^{-3}	634.2
7	184	899	20	10	68.23	1.000	76.76	46.440	1.24×10^{4}	10	8.09×10^{-4}	600.0
8	184	1641	10	100	51.22	1.000	55.83	33.630	3.59×10^{4}	10	2.78×10^{-4}	634.2
9	1296	1641	20	90	32.69	1.000	38.94	22.110	1.53×10^{5}	20	1.31×10^{-4}	366.2
10	1134	1641	20	70	27.31	1.000	34.49	19.800	2.34×10^{5}	20	8.53×10^{-5}	419.2
11	1641	2215	50	100	25.47	1.000	25.89	15.270	1.07×10^{6}	50	4.66×10^{-5}	547.0
12	1213	2215	20	50	22.34	1.000	25.3	14.930	1.31×10^{6}	20	1.53×10^{-5}	547.0
13	1630	2215	100	30	24.88	1.000	25.2	14.870	1.35×10^{6}	30	2.22×10^{-5}	547.0
14	1347	1630	20	70	16.71	1.000	17.12	9.798	3.72×10^{9}	20	5.38×10^{-9}	398.7
15	960	1630	20	50	13.54	1.000	13.95	8.405	7.76×10^{10}	20	2.58×10^{-10}	634.2
16	1595	1630	20	30	13.3	1.000	13.69	7.690	1.00×10^{11}	20	2.00×10^{-10}	299.4
17	1	1630	20	10	12.92	1.000	12.95	7.469	1.00×10^{11}	10	1.00×10^{-10}	450.0
18	1	1596	10	100	12.92	1.000	12.95	7.469	1.00×10^{11}	10	1.00×10^{-10}	450.0
19	1562	1596	20	90	2.829	1.000	0.2345	0.132	1.00×10^{11}	20	2.00×10^{-10}	299.4
										CUF total =	**0.1615**	

(b) A small third-world consulting company using the Unix philosophy (FeenoX+AWK+LaTeX)

Figure B.25.: Results of the same fatigue problem solved using two different philosophies.

```
$ feenox --version
FeenoX v0.3.292-gc932cb5
a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Copyright © 2009--2024 Seamplex, https://seamplex.com/feenox
GNU General Public License v3+, https://www.gnu.org/licenses/gpl.html.
FeenoX is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
$
```

The `--versions` option shows more information about the FeenoX build and the libraries the binary was linked against:

```
$ feenox -V
FeenoX v1.0.8-g731ca5d
a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Last commit date   : Wed Mar 20 08:11:05 2024 -0300
Build date         : Wed Mar 20 16:38:10 2024 -0300
Build architecture : linux-gnu x86_64
Compiler version   : gcc (Debian 12.2.0-14) 12.2.0
Compiler expansion : gcc -Wl,-z,relro -I/usr/include/x86_64-linux-gnu/mpich -L/usr/lib/x86_64- ↩
    linux-gnu -lmpich
Compiler flags     : -O3 -flto=auto -no-pie
Builder            : [redacted]@tom
GSL version        : 2.7.1
SUNDIALS version   : N/A
PETSc version      : Petsc Development GIT revision: v3.20.5-935-g78ad52f83fb  GIT Date:  ↩
    2024-03-25 05:31:58 +0000
PETSc arch         : arch-linux-c-debug
PETSc options      : --download-eigen --download-hdf5 --download-hypre --download-metis -- ↩
    download-mumps --download-parmetis --download-scalapack --download-slepc --with-64-bit- ↩
    indices=no --with-debugging=yes --with-precision=double --with-scalar-type=real PETSC_ARCH= ↩
    arch-linux-c-debug
SLEPc version      : SLEPc Development GIT revision: v3.20.1-36-g7a35a7b97  GIT Date: 2023-12-02 ↩
     02:30:03 -0600
$
```

The `--help` option gives a more detailed usage:

```
$ feenox --help
usage: feenox [options] inputfile [replacement arguments] [petsc options]

  -h, --help         display options and detailed explanations of commmand-line usage
  -v, --version      display brief version information and exit
  -V, --versions     display detailed version information
  -c, --check        validates if the input file is sane or not
  --pdes             list the types of PROBLEMs that FeenoX can solve, one per line
  --elements_info    output a document with information about the supported element types
  --linear           force FeenoX to solve the PDE problem as linear
  --non-linear       force FeenoX to solve the PDE problem as non-linear

  --progress         print ASCII progress bars when solving PDEs
  --mumps            ask PETSc to use the direct linear solver MUMPS
```

```
Instructions will be read from standard input if “”- is passed as
inputfile, i.e.

    $ echo 'PRINT 2+2' | feenox -
    4

The optional [replacement arguments] part of the command line mean that
each argument after the input file that does not start with an hyphen
will be expanded verbatim in the input file in each occurrence of $1,
$2, etc. For example

    $ echo 'PRINT $1+$2' | feenox - 3 4
    7

PETSc and SLEPc options can be passed in [petsc options] (or [options])
as well, with the difference that two hyphens have to be used instead of
only once. For example, to pass the PETSc option -ksp_view the actual
FeenoX invocation should be

    $ feenox input.fee --ksp_view

For PETSc options that take values, en equal sign has to be used:

    $ feenox input.fee --mg_levels_pc_type=sor

See https://www.seamplex.com/feenox/examples for annotated examples.

Report bugs at https://github.com/seamplex/feenox/issues
Ask questions at https://github.com/seamplex/feenox/discussions
Feenox home page: https://www.seamplex.com/feenox/
$
```

The input file provided as the first argument to the `feenox` binary contains all the information needed to solve the problem, so any further human intervention is not needed after execution begins, as requested by the SRS. If the execution finishes successfully, FeenoX returns a zero errorlevel to the calling shell (and follows the Unix *rule of silence*, i.e. no `PRINT` no output):

```
$ feenox maze.fee
$ echo $?
0
$
```

If there is problem during execution (including parsing and run-time errors), a line prefixed with `error:` is written into the standard error file descriptor and a non-zero errorlevel is returned:

```
$ feenox hello.fee
error: input file needs at least one more argument in commandline
$ echo $?
1
$ feenox hello.fee world
Hello world!
$ echo $?
0
```

```
$
```

This way, the error line can easily be parsed with standard Unix tools like `grep` and `cut` or with a proper regular expression parser. Eventually, any error should be forwarded back to the initiating entity—which depending on the workflow can be a human or an automation script—in order for her/him/it to fix it.

Following the *rule of repair* (section C.12), ill-defined input files with missing material properties or inconsistent boundary conditions are detected before the actual assembly of the matrix begins:

```
$ feenox thermal-1d-dirichlet-no-k.fee
error: undefined thermal conductivity 'k'
$ feenox thermal-1d-dirichlet-wrong-bc.fee
error: boundary condition 'xxx' does not have a physical group in mesh file 'slab.msh'
$
```

Error code are designed to be useful and helpful. An attempt to open a file might fail due to a wide variety of reasons. FeenoX clearly states which one caused the error so it can be remedied:

```
$ cat test.fee
READ_MESH cantilever.msh
$ feenox test.fee
$ chmod -r cantilever.msh
$ feenox test.fee
error: 'Permission denied' when opening file 'cantilever.msh' with mode 'r'
$ rm cantilever.msh
$ feenox test.fee
error: 'No such file or directory' when opening file 'cantilever.msh' with mode 'r'
$
```

If the command-line option `--progress` (or the `PROGRESS` keyword in `PROBLEM`) is used, then FeenoX writes into the standard output three "bars" showing the progress of

1. (.) the build and assembly of the problem matrices (stiffness and mass if applicable)
2. (-) the iterative solution of the problem (either linear or non-linear)
3. (=) the recovery of gradient-based (i.e. strains and stresses) out of the primary solution

```
$ gmsh -3 nafems-le10.geo
Info    : Running 'gmsh -3 nafems-le10.geo' [Gmsh 4.9.4-git-10d6a15fd, 1 node, max. 1 thread]
Info    : Started on Sat Feb  5 11:26:39 2022
Info    : Reading 'nafems-le10.geo'...
Info    : Reading 'nafems-le10.step'...
Info    :  - Label 'Shapes/Open CASCADE STEP translator 7.6 1' (3D)
Info    : Done reading 'nafems-le10.step'
Info    : Done reading 'nafems-le10.geo'
Info    : Meshing 1D...
[...]
Info    : Done optimizing mesh (0.106654 s)
Info    : Done optimizing high-order mesh (0.106654 s)
Info    : Done optimizing mesh (Wall 0.114461s, CPU 0.114465s)
Info    : 50580 nodes 40278 elements
Info    : Writing 'nafems-le10.msh'...
Info    : Done writing 'nafems-le10.msh'
```

```
Info    : Stopped on Sat Feb  5 11:26:40 2022 (From start: Wall 1.08693s, CPU 1.1709s)
$ feenox nafems-le10.fee --progress
................................................................................................
------------------------------------------------------------------------------------------------
================================================================================================
sigma_y @ D =   -5.38228        MPa
$
```

Once again, these ASCII-based progress bars can be parsed by the calling entity and then present it back to the user. For example, figure B.10 shows how the web-based GUI CAEplex shows progress inside an Onshape tab.

Since FeenoX uses PETSc (and SLEPc), command-line options can be passed from FeenoX to PETSc. The only difference is that since FeenoX follows the POSIX standard regarding options and PETSc does not, double dashes are required instead of PETSc' single-dash approach. That is to say, instead of `-ksp_monitor` one would have to pass `--ksp_monitor` (see section B.3.1.3 for details about the input files):

```
$ feenox thermal-1d-dirichlet-uniform-k.fee --ksp_monitor
  0 KSP Residual norm 1.913149816332e+00
  1 KSP Residual norm 2.897817223901e-02
  2 KSP Residual norm 3.059845525572e-03
  3 KSP Residual norm 1.943995979588e-04
  4 KSP Residual norm 7.418444674938e-06
  5 KSP Residual norm 1.233527903582e-07
0.5
$
```

Any PETSc command-line option takes precedence over the settings in the input file, so the preconditioner can be changed even if explicitly given with the `PRECONDITIONER` keyword:

```
$ feenox thermal-1d-dirichlet-uniform-k.fee --ksp_monitor --pc_type=ilu
  0 KSP Residual norm 2.678619047193e+00
  1 KSP Residual norm 7.172418823644e-16
0.5
$
```

If PETSc is compiled with MUMPS, FeenoX provides a `--mumps` option:

```
$ feenox thermal-1d-dirichlet-uniform-k.fee --ksp_monitor --mumps
  0 KSP Residual norm 1.004987562109e+01
  1 KSP Residual norm 4.699798436762e-15
0.5
$
```

An illustration of the usage of PETSc arguments and the fact that FeenoX automatically detects whether a problem is linear or not is given below. The case `thermal-1d-dirichlet-uniform-k.fee` ↩ is linear while the `two-squares.fee` from section section B.2.5 is not. Therefore, an SNES monitor should give output for the latter but not for the former. In effect:

```
$ feenox thermal-1d-dirichlet-uniform-k.fee --snes_monitor
0.5
```

```
$ feenox two-squares.fee --snes_monitor
  0 SNES Function norm 9.658033489479e+00
  1 SNES Function norm 1.616559951959e+00
  2 SNES Function norm 1.879821597500e-01
  3 SNES Function norm 2.972104258103e-02
  4 SNES Function norm 2.624028350822e-03
  5 SNES Function norm 1.823396478825e-04
  6 SNES Function norm 2.574514225532e-05
  7 SNES Function norm 2.511975376809e-06
  8 SNES Function norm 4.230090605033e-07
  9 SNES Function norm 5.154440365087e-08
$
```

As already explained in section B.2.2.2, FeenoX supports run-time replacement arguments that get replaced verbatim in the input file. This feature is handy when the same problem has to be solved over different meshes, such as when investigating the h-convergence order over Gmsh's element scale factor `-clscale`:

```
PROBLEM thermal 1D
READ_MESH slab-$1.msh
k(x) = 1+T(x)
BC left  T=0
BC right T=1
SOLVE_PROBLEM
PRINT nodes %+.2e integral((T(x)-(sqrt(1+(3*x))-1))^2,x,0,1)
```

```
$ for c in $(feenox steps.fee); do gmsh -v 0 -1 slab.geo -clscale ${c} -o slab-${c}.msh; feenox ↩
    thermal-1d-dirichlet-temperature-k-parametric.fee ${c}; done  | sort -g
11      +6.50e-07
13      +3.15e-07
14      +2.29e-07
15      +1.70e-07
17      +1.00e-07
20      +5.04e-08
24      +2.34e-08
32      +7.19e-09
39      +3.46e-09
49      +1.31e-09
$
```

Since the main input file is the first argument (not counting POSIX options starting with at least one dash), FeenoX might be invoked indirectly by adding a shebang line to the input file with the location of the system-wide executable and setting execution permissions on the input file itself. So if we modify the above `hello.fee` example as `hello`

```
#!/usr/local/bin/feenox
PRINT "Hello $1!"
```

and then we can do

```
$ chmod +x hello
$ ./hello world
Hello world!
```

```
$ ./hello universe
Hello universe!
$
```

For example, the following she-banged input file can be used to compute the derivative of a column with respect to the other as a Unix filter:

```
#!/usr/local/bin/feenox
FUNCTION f(t) FILE - INTERPOLATION steffen

a = vecmin(vec_f_t)
b = vecmax(vec_f_t)

# time step from arguments (or default 10 steps)
DEFAULT_ARGUMENT_VALUE 1 (b-a)/10
h = $1

VAR t'
f'(t) = derivative(f(t'),t',t)

PRINT_FUNCTION f' MIN a+0.5*h MAX b-0.5*h STEP h
```

```
$ feenox f.fee "sin(t)" 1 | ./derivative.fee
0.05    0.998725
0.15    0.989041
0.25    0.968288
0.35    0.939643
0.45    0.900427
0.55    0.852504
0.65    0.796311
0.75    0.731216
0.85    0.66018
0.95    0.574296
$
```

where `f.fee` is a "command-line function generator":

```
end_time = $2
PRINT t $1
```

B.3.1. Problem input

The problem should be completely defined by one or more input files. These input files might be

a. particularly formatted files to be read by the tool in an *ad-hoc* way, and/or
b. source files for interpreted languages which can call the tool through and API or equivalent method, and/or
c. any other method that can fulfill the requirements described so far.

Preferably, these input files should be plain ASCII files in order to allow to manage changes using distributed version control systems such as Git. If the tool provides an API for an interpreted

> language such as Python, then the Python source used to solve a particular problem should be Git-friendly. It is recommended not to track revisions of mesh data files but of the source input files, i.e. to track the mesher's input and not the mesher's output. Therefore, it is recommended not to mix the problem definition with the problem mesh data.
>
> It is not mandatory to include a GUI in the main distribution, but the input/output scheme should be such that graphical pre and post-processing tools can create the input files and read the output files so as to allow third parties to develop interfaces. It is recommended to design the workflow as to make it possible for the interfaces to be accessible from mobile devices and web browsers.
>
> It is expected that 80% of the problems need 20% of the functionality. It is acceptable if only basic usage can be achieved through the usage of graphical interfaces to ease basic usage at first. Complex problems involving non-trivial material properties and boundary conditions not be treated by a GUI and only available by needing access to the input files.

FeenoX currently works by reading an input file (which in turn can recursively `INCLUDE` further input files) with an *ad-hoc* format, whose rationale is described in this section. Therefore, it already does satisfy requirement a. but, eventually, could also satisfy requirement b. by adding a wrapper for high-level languages such as

- Python
- Julia
- R

that would either

i. create an input file and run FeenoX in the back, or
ii. successively call the FeenoX functions that define definitions and execute instructions (to be done).

As already explained in section B.1, the motto is "FeenoX is—in a certain sense—to desktop FEA programs and libraries what Markdown is to Word and (La)TeX, respectively and *deliberately*." Hence, the input files act as the Markdown source: instructions about what to do but not how to do it.

The input files are indeed plain-text ASCII files with English-like keywords that fully define the problem. The main features of the input format, thoroughly described below, are:

1. It is syntactically sugared by using English-like keywords.
2. Nouns are definitions and verbs are instructions.
3. Simple problems need simple inputs.
4. Simple things should be simple, complex things should be possible.
5. Whenever a numerical value is needed an expression can be given (i.e. "everything is an expression.")
6. The input file should match as much as possible the paper (or blackboard) formulation of the problem.
7. It provides means to compare numerical solutions against analytical ones.
8. It should be possible to read run-time arguments from the command line.
9. Input files are distributed version control-friendly.

B.3.1.1. Syntactic sugar & highlighting

The ultimate goal of FeenoX is to solve mathematical equations that are hard to solve with pencil and paper. In particular, to integrate differential equations (recall that the first usable computer was named ENIAC, which stands for Electronic Numerical Integrator and Computer). The input file format was designed as to how to ask the *computer* what to *compute*. The syntax, based on keywords and alphanumerical arguments was chosen as to sit in the middle of the purely binary numerical system employed by digital computers[6] and the purely linguistical nature of human communication. The rationale behind its design is that an average user can peek a FeenoX input file and tell what it is asking the computer to compute, as already illustrated for the NAFEMS LE10 problem in figure B.3. Even if the input files are created by a computer and not by a human, the code used to create a human-friendly input file will be human-friendlier than a code that writes only zeroes and ones as its output (that will become the input of another one following the Unix *rule of composition* section C.3).

The first argument not starting with a dash to the `feenox` executable is the path to the main input file. This main input file can in turn include other FeenoX input files (with the `INCLUDE` keyword) and/or read data from other files (such as meshes with the `READ_MESH` instruction) or other resources (such as data files for point-wise data interpolation with `FUNCTION` or shared memory objects TBD).

For instance, the test directory includes some spinning-disk cases that compare the analytical solution for the hoop and radial stresses with the numerical ones obtained with FeenoX. These cases read the radius R and thickness t from the `.geo` file used by Gmsh to build the mesh in the first place:

```
# analytical solution
INCLUDE spinning-disk-dimensions.geo
S_h(r) = ((3+nu)*R^2 - (1+3*nu)*r^2)
S_r(r) = (3+nu) * (R^2 - r^2)
```

where `spinning-disk-dimensions.geo` is

```
R = 0.1;
t = 0.003;
```

The input files are plain text files, either pure ASCII or UTF-8 (more details in section B.3.1.9). In principle any extension (even no extension) can be used for the FeenoX input files. Throughout the FeenoX repository and documentation the extension `.fee` is used, which has a couple of advantages:

1. The `.fee` extension is detected by syntax-highlighting extensions for common editors (both graphical such as Kate and cloud-friendly such as Vim) as illustrated in figure B.28.

2. The expression `$0` (or `${0}`) is expanded to the base name of the input file, i.e. the directory part (if present) is removed and the `.fee` extension is removed. Therefore,

   ```
   READ_MESH $0.msh
   ```

   would read a mesh file whose name is the same as the FeenoX input file, without the `.fee` extension.

[6] Analog and quantum computers are out of the scope.

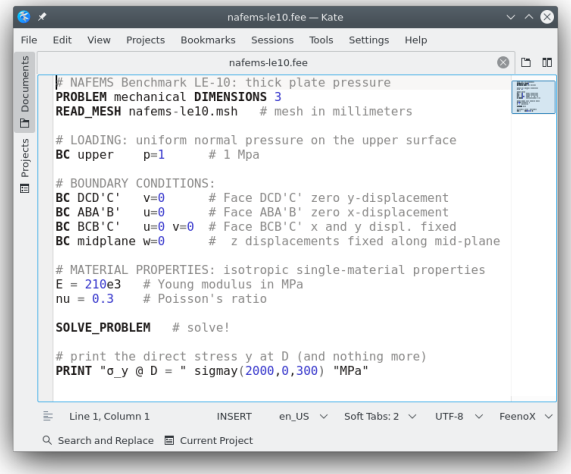


Figure B.26.: Kate

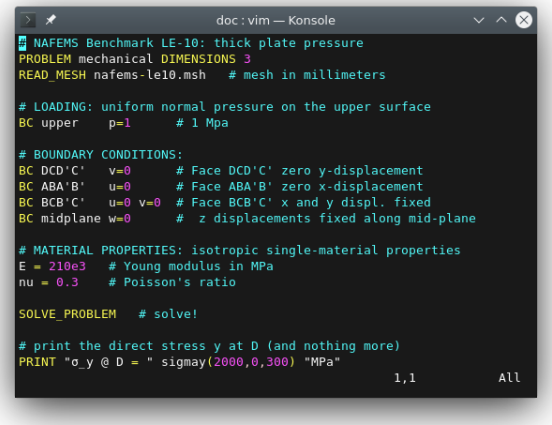


Figure B.27.: Vim

Figure B.28.: Syntax highlighting of input files in GUI and cloud-friendly text editors

B.3.1.2. Definitions and instructions

The way to tell the computer what problem it has to solve and how to solve it is by using keywords in the input file. Each non-commented line of the input file should start with either

i. a primary keyword such as `PROBLEM` or `READ_MESH`, or
ii. a variable such as `end_time` or a vector or matrix with the corresponding index(es) such as `v[2]` or `A[i][j]` followed by the `=` keyword, or
iii. a function name with its arguments such as `f(x,y)` followed by the `=` keyword.

A primary keyword usually is followed by arguments and/or secondary keywords, which in turn can take arguments as well. For example, in

```
PROBLEM mechanical DIMENSIONS 3
READ_MESH $0.msh
[...]
# print the direct stress y at D (and nothing more)
PRINT "σ_y @ D = " sigmay(2000,0,300) "MPa"
```

we have `PROBLEM` acting as a primary keyword, taking `mechanical` as its first argument and then `DIMENSIONS` as a secondary keyword with `3` being an argument to the secondary keyword. Then `READ_MESH` is another primary keyword taking `$0.msh` (which would be expanded to something like `nafems-le10.msh`) as its argument.

A primary keyword can be

1. a definition,
2. an instruction, or
3. both.

Definitions are English *nouns* and instructions are English *verbs*. In the example above, `PROBLEM` is a definition because it tells FeenoX about something it has to do (i.e. that it has to solve a three-dimensional problem), but does not do anything actually. On the other hand, `READ_MESH` is both a definition and an instruction: it defines that there exists a mesh named `nafems-le10.msh` which might be referenced later (for example in an `INTEGRATE` or `WRITE_MESH` instructions), but it also asks FeenoX to read the mesh at that point of the instruction list (more details below). Finally, `PRINT` is a primary keyword taking different types and number or arguments. It is an instruction because it does not define anything, it just asks FeenoX to print the value of the function named `sigmay` evaluated at $2000, 0, 300$. In this case, `sigmay` is a function which is implicitly defined when `PROBLEM` is set to `mechanical`. If `sigmay` was referenced before `PROBLEM`, FeenoX would not find it. And if the problem was of any other type, FeenoX would not find it even when referenced from the last line of the input file.

The following example further illustrates the differences between definitions and instructions. It compares the result of (numerically but adaptively) integrating $f(x,y,z) = \sin(x^3 + y^2 + z)$ over a unit cube against using a 3D Gauss integration scheme over a fixed set of quadrature points on the same unit cube meshes with two regular hexahedra in each direction (totaling 8 hexahedra). In one case hex20 are used and in the other one, hex27. Both cases use 27 quadrature points per element.

```
# these two are instructions to read a two meshes
# but they also define two mesh names that can be referred to later
READ_MESH hex20.msh DIM 3
READ_MESH hex27.msh DIM 3

# these three lines are definitions, they define three functions
# the first two also define four vectors for each function
#  1. vec_f20_x and vec_f27_x with the x coordinates of the mesh' nodes
#  2. vec_f20_y and vec_f27_y with the y coordinates of the mesh' nodes
#  3. vec_f20_z and vec_f27_z with the z coordinates of the mesh' nodes
#  4. vec_f20 and vec_f27 with the value of the function at the i-th node
# these definitions do not evaluate the functions, but they fill vectors 1-3
# (we'll fill vectors 4 below)
# note that these definitions refer to the meshes defined above in READ_MESH
FUNCTION f20(x,y,z) MESH hex20.msh
FUNCTION f27(x,y,z) MESH hex27.msh
f(x,y,z) = sin(x^3 + y^2 + z)

# these two lines are assignment instructions, they "fill" in
# the vectors with the value of the functinos f20(x,y,z) and f27(x,y,z)
# by evaluating f(x,y,z) at the nodes of the two meshes
# (there is a implicit loop for the index i over the size of the vectors)
vec_f20[i] = f(vec_f20_x[i], vec_f20_y[i], vec_f20_z[i])
vec_f27[i] = f(vec_f27_x[i], vec_f27_y[i], vec_f27_z[i])

# this line is an assignment, that first defines a variale If0
# and then calls the functional integral three times to perform a
# "continuous" (in the sense that it is numeric but adapative) triple integration
If0 = integral(integral(integral(f(x,y,z), z, 0, 1), y, 0, 1), x, 0, 1)

# these two lines are instructions, they integrate functions f20 and f27 over
# each of the meshes and then they store the results in the (implicitly defined)
# variables If20 and If27
INTEGRATE f20  MESH hex20.msh RESULT If20
INTEGRATE f27  MESH hex27.msh RESULT If27

# these lines are instructions, they print stuff to the standard output
```

```
# nothing is defined
PRINT %.10f If0
PRINT %.10f If20  %+.2e If20-If0
PRINT %.10f If27  %+.2e If27-If0
```

```
$ $ feenox  integral_over_hex.fee
0.7752945175
0.7753714586    +7.69e-05
0.7739155101    -1.38e-03
$
```

B.3.1.3. Simple inputs

Consider solving heat conduction on a one-dimensional slab spanning the unitary range $x \in [0,1]$. The slab has a uniform unitary conductivity $k = 1$ and Dirichlet boundary conditions

$$\begin{cases} T(0) &= 0 \\ T(1) &= 1 \end{cases}$$

This simple problem has the following simple input:

```
PROBLEM thermal 1D              # tell FeenoX what we want to solve
READ_MESH slab.msh              # read mesh in Gmsh's v4.1 format
k = 1                           # set uniform conductivity
BC left  T=0                    # set fixed temperatures as BCs
BC right T=1                    # "left" and "right" are defined in the mesh
SOLVE_PROBLEM                   # tell FeenoX we are ready to solve the problem
PRINT T(0.5)                    # ask for the temperature at x=0.5
```

```
$ feenox thermal-1d-dirichlet-uniform-k.fee
0.5
$
```

Now, if instead of having a uniform conductivity the problem had a space-dependent $k(x) = 1 + x$ then the input would read

```
PROBLEM thermal 1D
READ_MESH slab.msh
k(x) = 1+x                      # space-dependent conductivity
BC left  T=0
BC right T=1
SOLVE_PROBLEM
PRINT T(1/2) log(1+1/2)/log(2)  # print numerical and analytical solutions
```

```
$ feenox thermal-1d-dirichlet-space-k.fee
0.584893        0.584963
$
```

Finally, if the conductivity depended on temperature (rendering the problem non-linear) say like $k(x) = 1 + T(x)$ then

```
PROBLEM thermal 1D
READ_MESH slab.msh
k(x) = 1+T(x)                    # temperature-dependent conductivity
BC left  T=0
BC right T=1
SOLVE_PROBLEM
PRINT T(1/2) sqrt(1+(3*0.5))-1   # print numerical and analytical solutions
```

```
$ feenox thermal-1d-dirichlet-space-k.fee
0.581139        0.581139
$
```

Note that FeenoX could figure out by itself that the two first cases were linear while the last one was not. This can be verified by passing the extra argument `--snes_view`. In the first two cases, there will be no extra output. In the last one, the details of the non-linear solver used by PETSc will be written into the standard output. The experienced reader should take some time to compare the effort and level of complexity that other FEA solvers require in order to set up simple problems like these. A discussion of the difference between linear and non-linear problems can be found in the heat conduction tutorial.

B.3.1.4. Complex things

Alan Kay's idea that "simple things should be simple, complex things should be possible" has already been mentioned in section B.2.5. The first part of the quote was addressed in the previous section. Of course, complexity can scale up almost indefinitely so we cannot show an example right here. But the following repositories contain the scripts and input files (for Gmsh, FeenoX and other common Unix tools such as Sed and Awk) that solve non-trivial problems using FeenoX as the main tool and employing free and open source software only, both for the computation and for the creation of figures and result tables.

- Convergence study on stress linearization of an infinite pipe according to ASME: a parametric study over the number of elements through the thickness of a pipe and the error committed when computing membrane and bending stresses according to ASME VIII Div 2 Sec 5. The study uses the following element types

  - unstructured tet4
  - unstructured straight tet10
  - unstructured curved tet10
  - structured straight tet10
  - structured curved tet10
  - structured hex8
  - structured straight hex20
  - structured curved hex20
  - structured straight hex27
  - structured curved hex27

The linearized stresses for different number of elements through the pipe thickness are compared against the analytical solution. Figures with the meshes employed in each case and with plots of profiles vs. the radial coordinate and linearized stresses vs. number of elements through the thickness are created.

- Environmentally-assisted fatigue analysis of dissimilar material interfaces in piping systems of a nuclear power plant: a case that studies environmentally-assisted fatigue using environment factors applied to traditional in-air ASME fatigue analysis for operational an incidental transients in nuclear power plant as propose by EPRI. A fictitious CAD geometry representing a section of a piping system is studied. Four operational transients are made up with time-dependent data for pressure and water temperature.

  1. A transient heat conduction problem with temperature-dependent material properties (according to ASME property tables) are solved over a small region around a material interface between carbon and stainless steel.
  2. Primary stresses according to ASME are computed for each of the operational transients.
  3. The results of a modal analysis study are convoluted with a frequency spectrum of a design-basis earthquake using the SRSS method to obtain an equivalent static volumetric force distribution.
  4. The time-dependent temperature distribution for each transient is then used in quasi-static mechanical problems to compute secondary stresses according to ASME, including the equivalent seismic loads at the moment of higher thermal stresses.
  5. The history of linearized Tresca stresses are juxtaposed to compute the cumulative usage factors using the ASME peak-valley method.
  6. Environmental data is used to affect each cumulative usage factors with an environment factor to account for in-water conditions.

These repositories contain a `run.sh` that, when executed in a properly-set-up GNU/Linux host (either on premises or in the cloud), will perform a number of steps including

- creation of appropriate meshes
- execution FeenoX
- generation post-processing views, plots or tables with the results
- etc.

Refer to the READMEs in each repository for further details about the mathematical models involved.

B.3.1.5. Everything is an expression

As explained in the history of FeenoX (section D), the first objective of the code was to solve ODEs written in an ASCII file as human-friendly as possible. So even before any other feature, the first thing the FeenoX ancestors had was an algebraic parser and evaluator. This was back in 2009, and I performed an online search before writing the whole thing from scratch. I found a nice post in Slack Overflow[7] that discussed some cool ideas and even had some code published under the terms of the Creative Commons license.

[7] http://stackoverflow.com/questions/1384811/code-golf-mathematical-expression-evaluator-that-respects-pemdas

Besides ODEs, algebraic expressions are a must if one will be dealing with arbitrary functions in general and spatial distributions in particular—which is essentially what PDE solvers are for. If a piece of software wants to allow features ranging from comparing numerical results with analytical expression to converting material properties from a system of units to another one in a straightforward way for the user (either an actual human being or a computer creating an input file), it ought to be able to handle algebraic expressions.

Appropriately handling algebraic expressions leads to fulfilling the Unix rule of least surprise (section C.10). If the user needs to define a function $f(x) = 1/2 \cdot x^2$, all she has to do is write

```
f(x) = 1/2 * x^2
```

And conversely, if someone reads the line above, she can rest assure that there's a function called $f(x)$ that will evaluate to $1/2 \cdot x^2$ when needed. In effect, anyone can expect the output of this instruction

```
PRINT integral(f(x), x, 0, 1)
```

to be one third.

Of course, expressions are needed to get 100%-user defined output (further discussed in section B.3.2), as with the `PRINT` instruction above. But once an algebraic parser and evaluator is available, it does not make sense to keep force the user to write numerical data only. What if a the angular speed is in RPM and the model needs it in radians per second? Instead of having to write `104.72`, FeenoX allows the user to write

```
w = 1000 * 60*pi/180
```

This way,

1. it is easy to see what the speed in RPM is
2. precision is not lost
3. should the speed change, it is trivial to change the `1000` for anything else.

Whenever an input keyword needs a numerical value, any expression will do:

```
n = 3
VECTOR x SIZE 2+n
x[i] = i^2
PRINT x
```

```
$ feenox vector_size_one_plus_n.fee
1       4       9       16      25
$
```

It goes without saying that algebraic expressions are a must when defining transient and/or space-dependent boundary conditions for PDEs:

```
PROBLEM thermal 1D
READ_MESH slab.msh

end_time = 10
k = 1
kappa = 0.1

FUNCTION f(t) DATA {
0    0
1    1
2    1
3    2
4    0
10   0
}
BC left  T=f(t)

w = 0.5*pi
BC right T=1+sin(w*t)

SOLVE_PROBLEM
PRINT t T(0) T(0.5) T(1)
```

Besides purely algebraic expressions, FeenoX can handle point-wise defined functions which can then be used in algebraic expressions. A useful example is allowing material properties (e.g. Young modulus) to depend on temperature. Consider a simple plane-strain square $[-1,+1]\times[-1,+1]$ fixed on one side and with a uniform tension in the opposite one while leaving the other two free. The square's Young modulus depends on temperature according to a one-dimensional point-wise defined function $E_{\text{carbon}}(T)$ given by pairs stated according to one of the many material properties tables from ASME II and interpolated using Steffen's method. Although in this example the temperature is given as an algebraic expression of space, in particular

$$T(x,y)\ [^{\circ}\text{C}] = 200 + 350\cdot y$$

```
PROBLEM mechanical plane_strain
READ_MESH square-centered.msh # [-1:+1]x[-1:+1]

# fixed at left, uniform traction in the x direction at right
BC left    fixed
BC right   tx=50

# ASME II Part D pag. 785 Carbon steels with C<=0.30%
FUNCTION E_carbon(temp) INTERPOLATION steffen DATA {
-200  216
-125  212
-75   209
25    202
100   198
150   195
200   192
250   189
300   185
350   179
400   171
450   162
```

```
500   151
550   137
}


# known temperature distribution
# (we could have read it from an outpout of a thermal problem)
T(x,y) := 200 + 350*y

# Young modulus is the function above evaluated at the local temperature
E(x,y) := E_carbon(T(x,y))

# uniform Poisson's ratio
nu = 0.3

SOLVE_PROBLEM
WRITE_MESH mechanical-square-temperature.vtk  E  VECTOR u v 0
```

By replacing `T(x,y)= 200 + 350*y` with `T(x,y)= 200` we can compare the results of the temperature-dependent case with the uniform-properties case (figure B.31):

```
$ feenox mechanical-square-temperature.fee
$ diff mechanical-square-temperature.fee mechanical-square-uniform.fee
29c29
< T(x,y) := 200 + 350*y
---
> T(x,y) := 200
38c38
< WRITE_MESH mechanical-square-temperature.vtk  E  VECTOR u v 0
---
> WRITE_MESH mechanical-square-uniform.vtk  E  VECTOR u v 0
$ feenox mechanical-square-uniform.fee
$
```

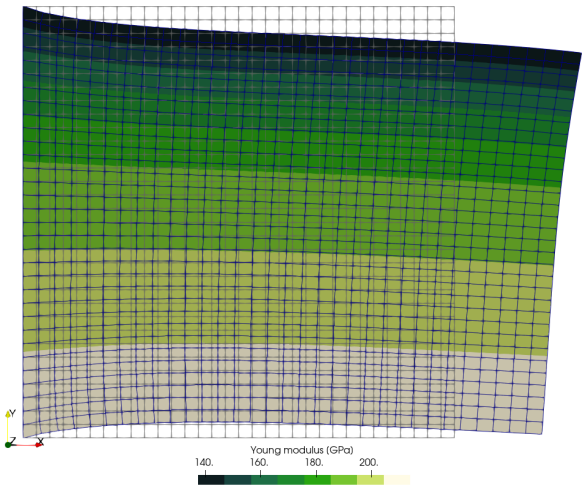


Figure B.29.: Temperature-dependent E

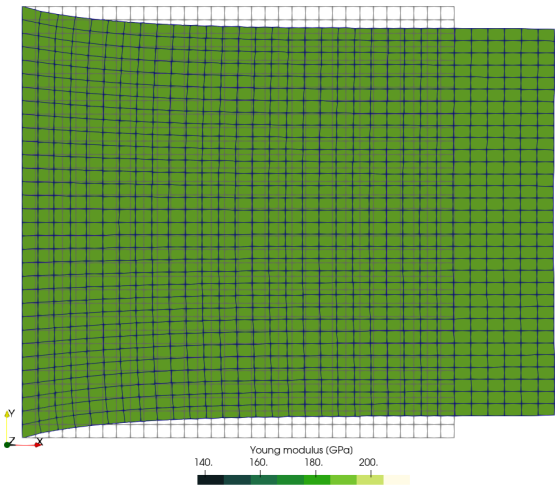


Figure B.30.: Uniform E for reference

Figure B.31.: Mechanical plane-strain square with temperature-dependent Young modulus and comparison with uniform reference case.

In real applications this distribution $T(x, y)$ can be read from the output of an actual heat conduction problem. See section B.3.2.2 for a revisit of this case, reading the temperature from an unstructured triangular mesh instead of hard-coding it as an algebraic expression of space.

So remember, in FeenoX *everything is an expression*—especially temperature, as also shown in the next section.

B.3.1.6. Matching formulations

The Lorenz' dynamical system system and the NAFEMS LE10 benchmark problem, both discussed earlier in section B.1.2, illustrate how well the FeenoX input file matches the usual human-readable formulation of ODE or PDE problems. In effect, figure B.3 shows there is a trivial one-to-one correspondence between the sections of the problem formulated in a sheet of paper and the text file `nafems-le10.fee`. The same effect can be seen in the NAFEMS LE11 problem, which involves a temperature distribution given as an algebraic expression of **x**:

Let us consider the NAFEMS LE11 benchmark problem titled "Solid cylinder/taper/sphere-temperature" stated in figure B.32. It consists of an axi-symmetrical geometry subject to thermal loading by a temperature distribution given by an algebraic expression. The material properties are linear, orthotropic and uniform. The boundary conditions prescribe symmetries in all directions.

- Loading
  - Linear temperature gradient in the radial an axial direction

$$T(x, y, z)\ [^{\circ}\mathrm{C}] = \left(x^2 + y^2\right)^{1/2} + z$$

- Boundary conditions
  - Symmetry on x-z plane, i.e. zero y-displacement
  - Symmetry on y-z plane, i.e. zero x-displacement
  - Face on x-y plane zero z-displacement
  - Face $HIH'I'$ zero z-displacement
- Material properties
  - Isotropic, $E = 210 \times 10^3$ MPa, $\nu = 0.3$
  - Thermal expansion coefficient $\alpha = 2.3 \times 10^{-4}\ ^{\circ}\mathrm{C}^{-1}$
- Output
  - Direct stress σ_{zz} at point A

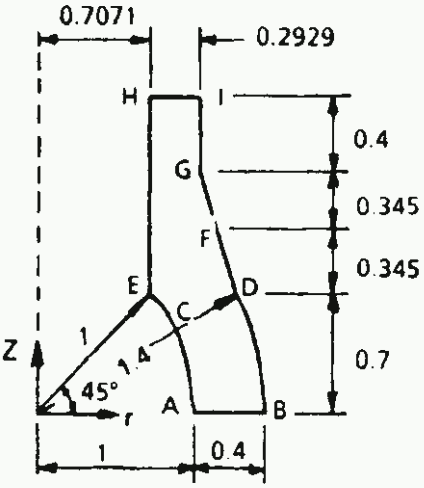
To solve this problem, we can use the following FeenoX input file that exactly matches the human-readable formulation:

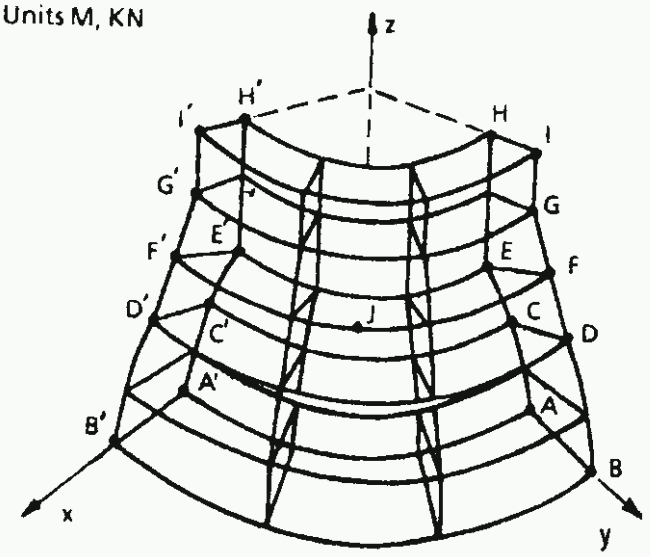
```
PROBLEM mechanical
READ_MESH $0.msh

# linear temperature gradient in the radial and axial direction
T(x,y,z) = (x^2 + y^2)^(1/2) + z
```

NAFEMS	SOLID CYLINDER / TAPER / SPHERE - TEMPERATURE	Test No LE11	DATE / ISSUE 15 - 6 - 90/2

ORIGIN	NAFEMS report LSB2
ANALYSIS TYPE	Linear elastic solid

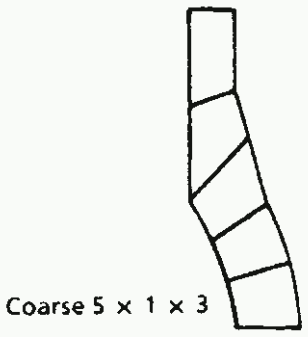
GEOMETRY — Units M, KN



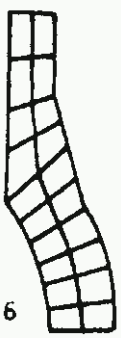


LOADING	Linear temperature gradient in the radial and axial direction $T\,°C = (x^2 + y^2)^{1/2} + z$
BOUNDARY CONDITIONS	Symmetry on xz-plane i.e. zero y-displacement Symmetry on yz-plane i.e. zero x-displacement Face on xy-plane zero z-displacement Face HIH'I' zero z-displacement
MATERIAL PROPERTIES	Isotropic, $E = 210 \times 10^3$ MPa, $\nu = 0.3$ $\alpha = 2.3 \times 10^{-4}$ / °C
ELEMENT TYPES	Solid hexahedra, wedges and tetrahedra

MESHES



Coarse 5 × 1 × 3



Fine 10 × 2 × 6

OUTPUT — Direct stress σ_{zz} at point A	TARGET -105 MPa (refined axisymmetric)

Figure B.32.: Formulation of the NAFEMS LE11 problem.

```
# Boundary conditions
BC xz     symmetry
BC yz     symmetry
BC xy     w=0
BC HIH'I' w=0

# material properties (isotropic & uniform so we can use scalar constants)
E = 210e3*1e6       # mesh is in meters, so E=210e3 MPa -> Pa
nu = 0.3            # dimensionless
alpha = 2.3e-4      # in 1/°C as in the problem

SOLVE_PROBLEM
WRITE_RESULTS FORMAT vtk
PRINT "sigma_z(A) =" sigmaz(0,1,0)/1e6 "MPa (target was -105 MPa)" SEP " "
```

```
$ time feenox nafems-le11.fee
sigma_z(A) = -105.041 MPa (target was -105 MPa)

real    0m1.766s
user    0m1.642s
sys     0m0.125s
```

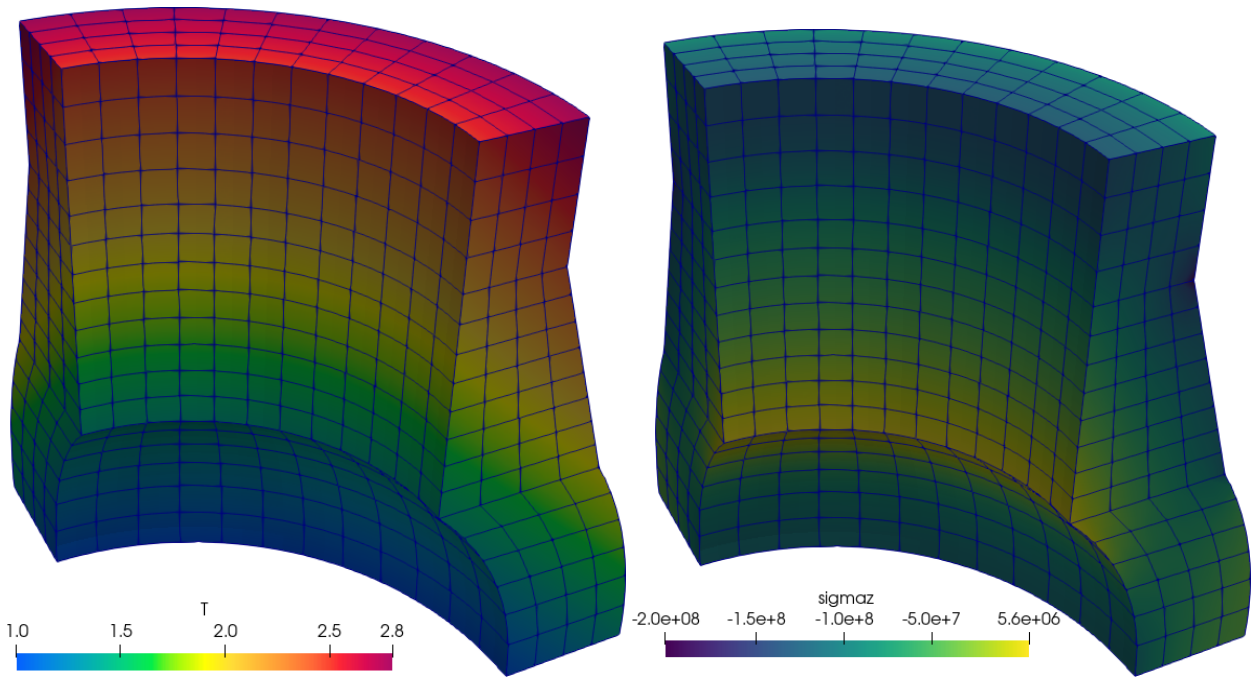


Figure B.33.: The NAFEMS LE11 problem results

B.3.1.7. Comparison of solutions

One cornerstone design feature is that FeenoX has to provide a way to compare its numerical results with other already-know solutions—either analytical or numerical—in order to verify their validity. Indeed, being able to take the difference between the numerical result and an algebraic expression evaluated at arbitrary locations (usually quadrature points to compute~L_p norms of the error) is a must if code verification is required.

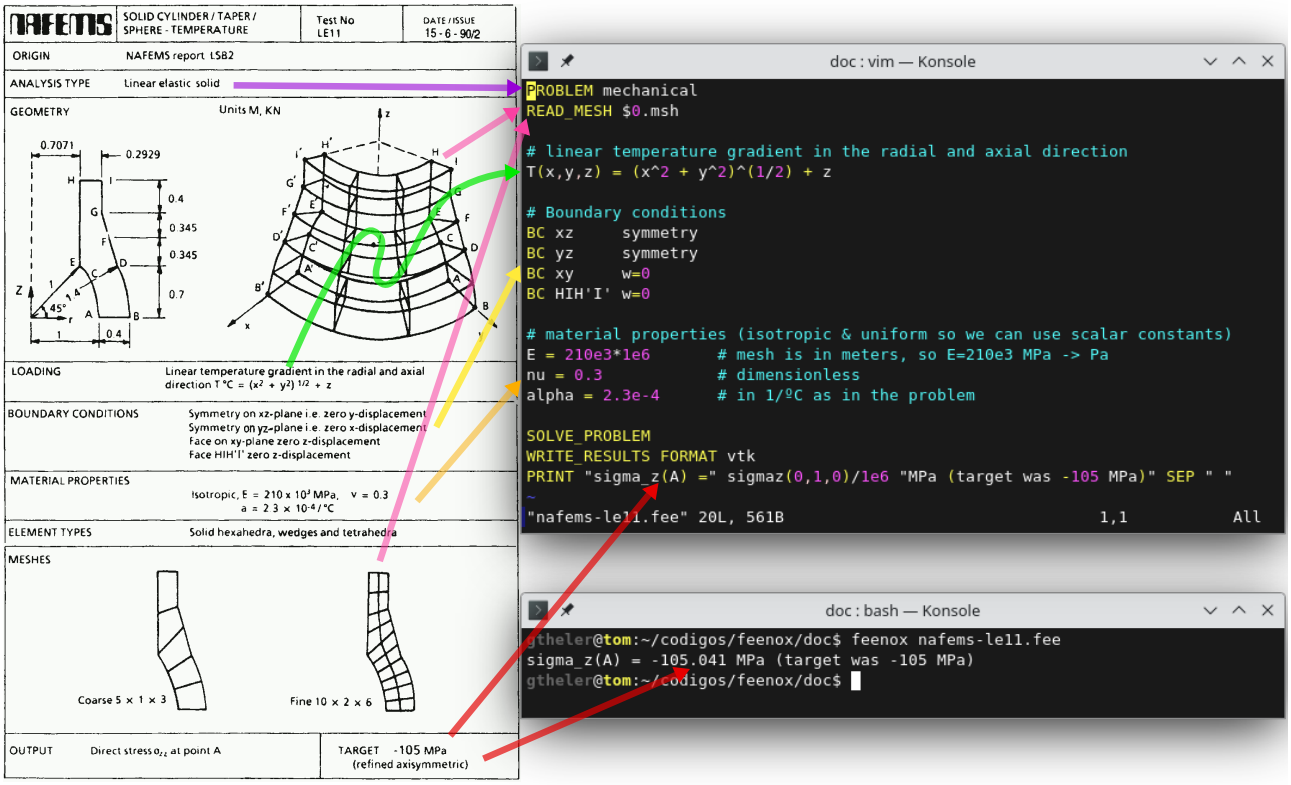


Figure B.34.: The NAFEMS LE11 problem statement and the corresponding FeenoX input

Let us consider a one-dimensional slab reactor with uniform macroscopic cross sections

$$\begin{aligned}\Sigma_t &= 0.54628\ \text{cm}^{-1}\\ \Sigma_s &= 0.464338\ \text{cm}^{-1}\\ \nu\Sigma_f &= 1.70 \cdot 0.054628\ \text{cm}^{-1}\end{aligned}$$

such that, if computed with neutron transport theory, is exactly critical with a width of $a = 2 \cdot 10.371065$ cm. Just to illustrate a simple case, we can solve it using the diffusion approximation with zero flux at both as. This case has an analytical solution for both the effective multiplication factor

$$k_{\text{eff}} = \frac{\nu\Sigma_f}{(\Sigma_t - \Sigma_s) + D \cdot \left(\frac{\pi}{a}\right)^2}$$

and the flux distribution

$$\phi(x) = \frac{\pi}{2} \cdot \sin\left(\frac{x}{a} \cdot \pi\right)$$

provided the diffusion coefficient D is defined as

$$D = \frac{1}{3 \cdot \Sigma_t}$$

We can solve both the numerical and analytical problems in FeenoX, and more importantly, we can subtract them at any point of space we want:

```
PROBLEM neutron_diffusion 1D
READ_MESH slab-UD20-1-0-SL.msh

a = 2 * 10.371065 # critical size of the problem UD20-1-0-SL (number 22 report Los Alamos)

Sigma_t1 = 0.54628
Sigma_s1.1 = 0.464338
nuSigma_f1 = 1.70*0.054628
D1 = 1/(3*Sigma_t1)

# null scalar flux at both ends of the slab
BC left  null
BC right null

SOLVE_PROBLEM

# analytical effective multiplication factor (diffusion approximation)
keff_diff = nuSigma_f1/((Sigma_t1-Sigma_s1.1) + D1*(pi/a)^2)

# analytical normalized flux distribution (diffusion approximation)
phi_diff(x) = pi/2 * sin(x/a * pi)

PRINT_FUNCTION FORMAT %+.3f phi1 phi_diff phi1(x)-phi_diff(x) HEADER
PRINT TEXT "\# keff      = " %.8f keff
PRINT TEXT "\# kdiff     = " %.8f keff_diff
PRINT TEXT "\# rel error = " %+.2e (keff-keff_diff)/keff
```

```
$ feenox neutron-diffusion-1d-null.fee
# x     phi1    phi_diff        phi1(x)-phi_diff(x)
+0.000  +0.000  +0.000  +0.000
+10.371 +1.574  +1.571  +0.003
+20.742 +0.000  +0.000  -0.000
+1.474  +0.348  +0.348  +0.001
+2.829  +0.654  +0.653  +0.001
+4.074  +0.911  +0.909  +0.002
+5.217  +1.118  +1.116  +0.002
+6.268  +1.280  +1.277  +0.002
+7.233  +1.399  +1.397  +0.003
+8.120  +1.483  +1.480  +0.003
+8.935  +1.537  +1.534  +0.003
+9.683  +1.565  +1.562  +0.003
+11.059 +1.565  +1.562  +0.003
+11.807 +1.537  +1.534  +0.003
+12.622 +1.483  +1.480  +0.003
+13.509 +1.399  +1.397  +0.003
+14.474 +1.280  +1.277  +0.002
+15.525 +1.118  +1.116  +0.002
+16.668 +0.911  +0.909  +0.002
+17.913 +0.654  +0.653  +0.001
+19.268 +0.348  +0.348  +0.001
```

```
# keff      =   0.96774162
# kdiff     =   0.96797891
# rel error =   -2.45e-04
$
```

Something similar could have been done for two groups of energy, although the equations get a little bit more complex so we leave it as an example in the Git repository.

A notable non-trivial thermo-mechanical problem that nevertheless has an analytical solution for the displacement field is the "Parallelepiped whose Young's modulus is a function of the temperature" (figure B.35). The problem consists of finding the non-dimensional temperature T and displacements u, v and w distributions within a solid parallelepiped of length ℓ whose base is a square of $h \times h$. The solid is subject to heat fluxes and to a traction pressure at the same time. The non-dimensional Young's modulus E of the material depends on the temperature T in a know algebraically way, whilst both the Poisson coefficient ν and the thermal conductivity k are uniform and do not depend on the spatial coordinates:

$$E(T) = \frac{1000}{800 - T}$$
$$\nu = 0.3$$
$$k = 1$$

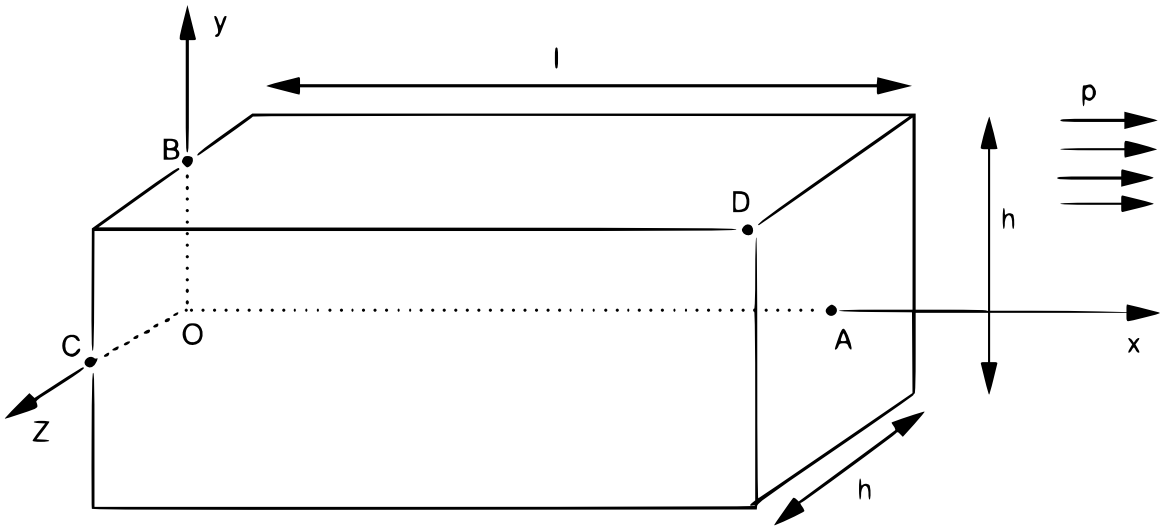


Figure B.35.: Parallelepiped whose Young's modulus is a function of the temperature. Original figure from v7.03.100.pdf

The thermal boundary conditions are

- Temperature at point A at $(\ell, 0, 0)$ is zero
- Heat flux q'' through $x = \ell$ is +2
- Heat flux q'' through $x = 0$ is -2
- Heat flux q'' through $y = h/2$ is +3

- Heat flux q'' through $y = -h/2$ is -3
- Heat flux q'' through $z = h/2$ is +4
- Heat flux q'' through $z = -h/2$ is -4

The mechanical boundary conditions are

- Point O at $(0, 0, 0)$ is fixed
- Point B at $(0, h/2, 0)$ is restricted to move only in the y direction
- Point C at $(0, 0, /h2)$ cannot move in the x direction
- Surfaces $x = 0$ and $x = \ell$ are subject to an uniform normal traction equal to one

The analytical solution is

$$
\begin{aligned}
T(x,y,z) &= -2x - 3y - 4z + 40 \\
u(x,y,z) &= \frac{A}{2} \cdot [x^2 + \nu \cdot (y^2 + z^2)] + B \cdot xy + C \cdot xz + D \cdot x - \nu \cdot \frac{Ah}{4} \cdot (y + z) \\
v(x,y,z) &= -\nu \cdot \left[A \cdot xy + \frac{B}{2} \cdot \left(y^2 - z^2 + \frac{x^2}{\nu} \right) + C \cdot yz + D \cdot y - A \cdot h/4 \cdot x - C \cdot h/4 \cdot z \right] \\
w(x,y,z) &= -\nu \cdot \left[A \cdot xz + B \cdot yz + C/2 \cdot \left(z^2 - y^2 + \frac{x^2}{\nu} \right) + D \cdot z + \frac{Ch}{4} \cdot y - \frac{Ah}{4} \cdot x \right]
\end{aligned}
$$

where~$A = 0.002$, $B = 0.003$, $C = 0.004$ and~$D = 0.76$. The reference results are the temperature at points O and D and the displacements at points A and D (tabla B.5}.

Point	Unknown	Reference value
O	T	+40.0
D	T	-35.0
A	u	+15.6
	v	-0.57
	w	-0.77
D	u	+16.3
	v	-1.785
	w	-2.0075

Tabla B.5.: Reference results the original benchmark problem

First, the thermal problem is solved with FeenoX and the temperature distribution $T(x, y, z)$ is written into a `.msh` file.

```
PROBLEM neutron_diffusion 1D
READ_MESH slab-UD20-1-0-SL.msh

a = 2 * 10.371065 # critical size of the problem UD20-1-0-SL (number 22 report Los Alamos)

Sigma_t1 = 0.54628
Sigma_s1.1 = 0.464338
nuSigma_f1 = 1.70*0.054628
```

```
D1 = 1/(3*Sigma_t1)

# null scalar flux at both ends of the slab
BC left  null
BC right null

SOLVE_PROBLEM

# analytical effective multiplication factor (diffusion approximation)
keff_diff = nuSigma_f1/((Sigma_t1-Sigma_s1.1) + D1*(pi/a)^2)

# analytical normalized flux distribution (diffusion approximation)
phi_diff(x) = pi/2 * sin(x/a * pi)

PRINT_FUNCTION FORMAT %+.3f phi1 phi_diff phi1(x)-phi_diff(x) HEADER
PRINT TEXT "\# keff      = " %.8f keff
PRINT TEXT "\# kdiff     = " %.8f keff_diff
PRINT TEXT "\# rel error = " %+.2e (keff-keff_diff)/keff
```

Then, the mechanical problem reads two meshes: one for solving the actual mechanical problem and another one for reading $T(x, y, z)$ from the previous step. Note that the former contains second-order hexahedra and the latter first-order tetrahedra. After effectively solving the problem, it writes again tabla B.5 in Markdown.

B.3.1.8. Run-time arguments

The usage of run-time command-line arguments was illustrated in section B.2.2.2. The idea is that if the expression `$n` (or `${n}`) is found in the input file, the FeenoX parser expands the expression literally as the n-th non-optional argument in the command line. The case $n = 0$ is particular in the sense that, as explained in section B.3.1.1, expands to the name of the input file without the leading directory path and the trailing extension `.fee`.

The definition `DEFAULT_ARGUMENT_VALUE` can be used to give a default value for arguments not provided. otherwise, FeenoX would complain:

```
$ echo "PRINT \$1" | feenox -
error: input file needs at least one more argument in commandline
$ echo -e "DEFAULT_ARGUMENT_VALUE 1 hello\nPRINT \$1" | feenox -
hello
$
```

This feature is extensively used in parametric and optimization runs such as in the verification using the Method of Manufactured solutions:

```
# MMS data, set T_mms(x) and k_mms(x) as desired
T_mms(x,y) = 1 + sin(2*x)^2 * cos(3*y)^2
k_mms(x,y) = 1 + x - 0.5*y

READ_MESH square-$2-$3-$4.msh DIMENSIONS 2
PROBLEM thermal

DEFAULT_ARGUMENT_VALUE 1 dirichlet # BCs = dirichlet/neumann
DEFAULT_ARGUMENT_VALUE 2 tri3      # shape = tri3/tri6/quad4/quad8/quad9
DEFAULT_ARGUMENT_VALUE 3 struct    # algorithm = struct/frontal/delaunay
```

```
DEFAULT_ARGUMENT_VALUE 4 8          # refinement factor = 1/2/3/4...
DEFAULT_ARGUMENT_VALUE 5 0          # write vtk? = 0/1

# read the results of the symbolic derivatives
INCLUDE thermal-square-q.fee

# set the PDE coefficients and BCs we just read above
k(x,y) = k_mms(x,y)
q(x,y) = q_mms(x,y)

# set the BCs (depending on $1)
INCLUDE thermal-square-bc-$1.fee

SOLVE_PROBLEM   # this line should be self-explanatory

# output
PHYSICAL_GROUP bulk DIM 2
h = sqrt(bulk_area/cells)

# L-2 error
INTEGRATE (T(x,y)-T_mms(x,y))^2 RESULT e_2
error_2 = sqrt(e_2)

# L-inf error
FIND_EXTREMA abs(T(x,y)-T_mms(x,y)) MAX error_inf

PRINT %.6f log(h) log(error_inf) log(error_2) %g $4 cells nodes %.2f 1024*memory() wall_time()

IF $5
  WRITE_MESH thermal-square_$1-$2-$3-$4.vtk T q T_mms T(x,y)-T_mms(x,y)
ENDIF
```

which is called from a Bash loop that looks like

```
bcs="dirichlet neumann"
elems="tri3 tri6 quad4 quad8 quad9"
algos="struct frontal delaunay"
cs="4 6 8 10 12 16 20 24 30 36 48"

[...]

for bc in ${bcs}; do
 for elem in ${elems}; do
  for algo in ${algos}; do

    [...]

    for c in ${cs}; do

     name="thermal_square_${bc}-${elem}-${algo}-${c}"

     # prepare mesh
     if [ ! -e square-${elem}-${algo}-${c}.msh ]; then
       lc=$(echo "PRINT 1/${c}" | feenox -)
       gmsh -v 0 -2 square.geo ${elem}.geo ${algo}.geo -clscale ${lc} -o ↩
           square-${elem}-${algo}-${c}.msh
     fi

     # run feenox
     feenox thermal-square.fee ${bc} ${elem} ${algo} ${c} ${vtk} | tee -a ${dat}.dat
```

```
    done

    [...]

  done
 done
done
```

The full script can be found in `tests/mms/thermal2d/2d/run.sh`.

In the input file above, the instruction `WRITE_MESH` with an explicit file name was given

```
WRITE_MESH thermal-square_$1-$2-$3-$4.vtk T q T_mms T(x,y)-T_mms(x,y)
```

because non-standard output fields are needed (namely `T_mms` and `T(x,y)-T_mms(x,y)`). If the `WRITE_RESULTS` is used without and explicit `FILE` keyword, the output file name is the basename of the input file and the expansion of all the arguments in the command line, i.e. `$0-[$1-[$2...]].msh`.

The study "Comparison of resource consumption for different FEA programs" also performs a parametric run on the mesh size using similar ideas.

B.3.1.9. Git and macro-friendliness

The FeenoX input files as plain ASCII files by design. This means that they can be tracked with Git or any other version control system so as to allow reliable traceability of computations. Along with the facts that FeenoX interacts well with

a. Gmsh, that can either use ASCII input files as well or be used as an API from C, C++, Python and Julia, and
b. Other scripting languages such as Bash, Python or even AWK, whose input files are ASCII files as well,

makes it possible to track a whole computation using FeenoX as a Git repository, as already exemplified in section B.3.1.4. It is important to note that what files that should be tracked in Git include

1. READMEs and documentation in Markdown
2. Shell scripts
3. Gmsh input files and/or scripts
4. FeenoX input files

Files that should not be tracked include

1. Documentation in HTML or PDF
2. Mesh files
3. VTK and result files

since in principle they could be generated from the files in the Git repository by executing the scripts, Gmsh and/or FeenoX.

Even more, in some cases, the FeenoX input files—following the Unix rule of generation section C.14 be created out of generic macros that create particular cases. For example, say one has a mesh of a fin-based dissipator where all the surfaces are named `surf_1_`i for $i = 1, \ldots, 26$. All of them will have a convection boundary condition while surface number 6 is the one that is attached to the electronic part that has to be cooled. Instead of having to "manually" giving the list of surfaces that have the convection condition, we can use M4 to do it for us:

```
PROBLEM thermal 3d
READ_MESH fins.msh

include(forloop.m4)
BC convection h=10 Tref=-5 forloop(i, 1, 5, `PHYSICAL_GROUP "surf_1_``'i"' ) forloop(i, 7, 26,  ↩
    `PHYSICAL_GROUP "surf_1_``'i"' )

BC surf_1_6 q=1000
k = 237
SOLVE_PROBLEM
WRITE_MESH fins.vtk T
```

Note that since FeenoX was born in Unix, we can pipe the output of `m4` to FeenoX directly by using `-` as the input file in the command line:

```
$ m4 fins.fee.m4 | feenox -
$
```

Figura B.36 confirms that all the faces have the right boundary conditions: face number six got the power BC and all the rest got the convection BC.

Besides being ASCII files, should special characters be needed for any reason within a particular application of FeenoX, UTF-8 characters can be used natively as illustrated in figure B.39.

B.3.2. Results output

> The output ought to contain useful results and should not be cluttered up with non-mandatory information such as ASCII art, notices, explanations or copyright notices. Since the time of cognizant engineers is far more expensive than CPU time, output should be easily interpreted by either a human or, even better, by other programs or interfaces—especially those based in mobile and/or web platforms. Open-source formats and standards should be preferred over privative and ad-hoc formatting to encourage the possibility of using different workflows and/or interfaces.

The output in FeenoX is 100% user defined, i.e. everything that FeenoX writes comes from one of the following output instructions:

- PRINT
- PRINTF
- PRINT_FUNCTION

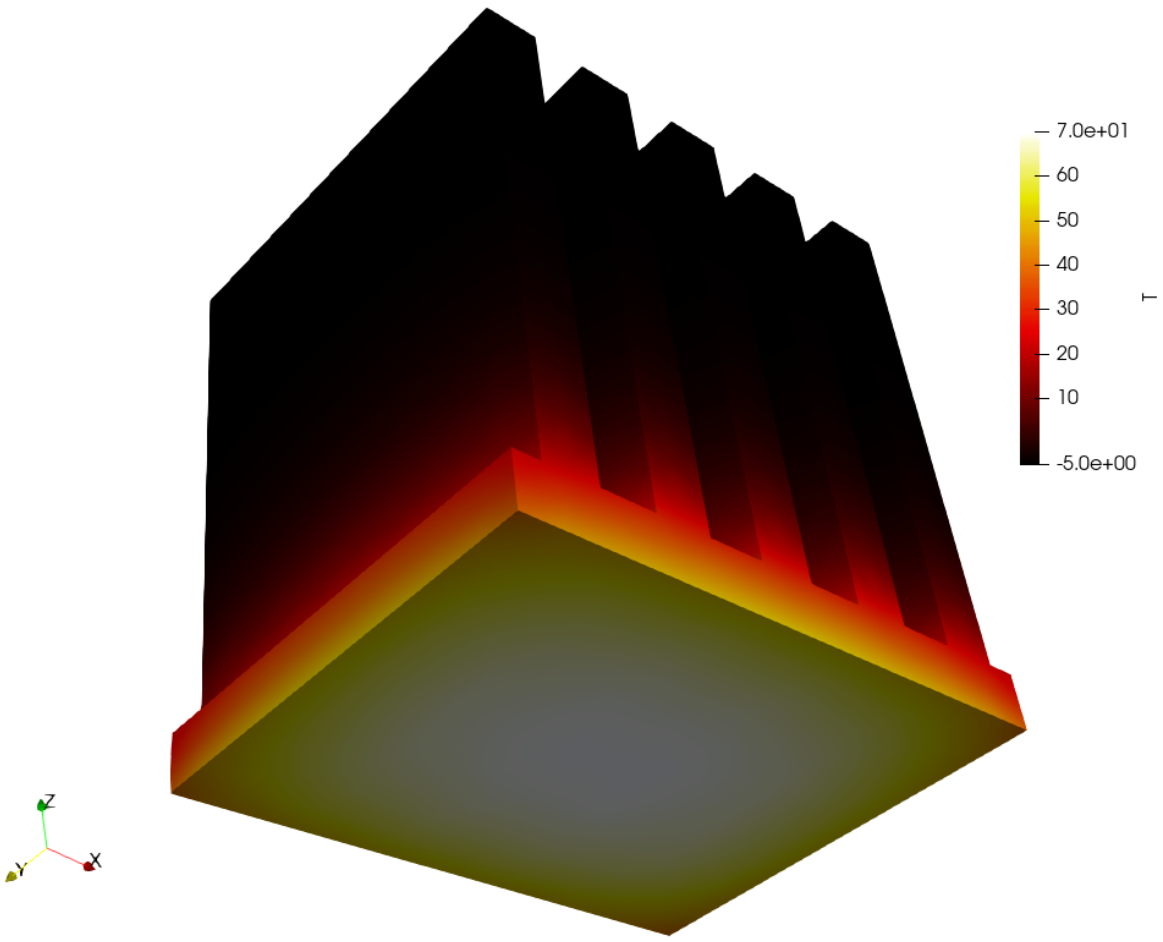


Figure B.36.: Temperature distribution in a fin dissipator where all the faces have a convection BC except one that has a fixed heat flux of $q'' = 1,000\text{W} \cdot \text{m}^{-2}$.

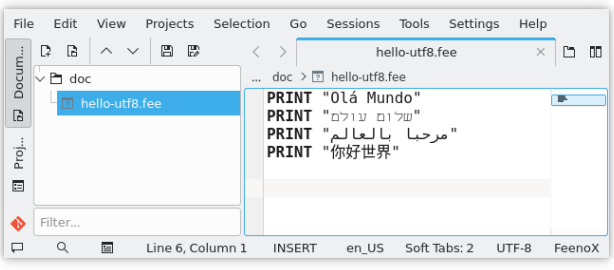


Figure B.37.: UTF-8 in Kate

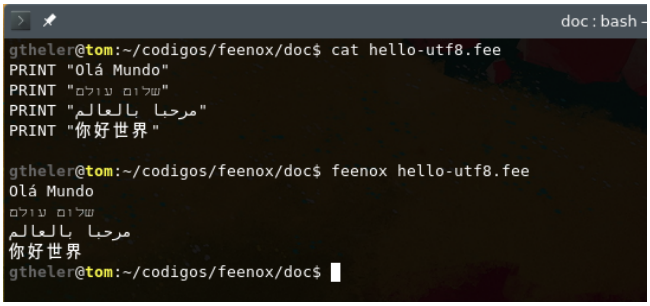


Figure B.38.: UTF-8 in Bash (through Konsole)

Figure B.39.: Special characters in Kate and in Bash.

- PRINT_VECTOR
- WRITE_MESH
- WRITE_RESULTS
- DUMP

In the absence of any of these instructions, FeenoX *will not* write anything. Not in the standard output, not in any other file. Nothing (Unix rule of silence, section C.11).

Computer	Monthly Rental	Relative Speed	First Delivery
CDC 3800	$ 50,000	1	Jan 66
CDC 6600	$ 80,000	6	Sep 64
CDC 6800	$ 85,000	20	Jul 67
GE 635	$ 55,000	1	Nov 64
IBM 360/62	$ 58,000	1	Nov 65
IBM 360/70	$ 80,000	2	Nov 65
IBM 360/92	$ 142,000	20	Nov 66
PHILCO 213	$ 78,000	2	Sep 65
UNIVAC 1108	$ 45,000	2	Aug 65

Tabla B.6.: Relative speed is expressed with reference to IBM 7030. Data for computers expected to appear after 1965 was estimated.

This is a sound design decision that follows the Unix rules of silence and, more importantly, of economy. In effect, more than fifty years ago CPU time was far more expensive than engineering time (tabla B.6). At that time, engineering programs had to write *everything* they computed because it was too expensive to re-run the calculation in case a single result was missing.

Nowadays the engineering time is far more expensive than CPU time. Therefore, the time needed for the user to find and process a single result in a soup of megabytes of a cluttered output file far outweighs the cost of running a computation from scratch with the needed result as the only output. Especially if the expensive engineers are smart enough to set up the problem using a coarse mesh and run the actual fine execution only after having checked everything works as expected.

The input file from the tensile-test tutorial illustrates this idea: only 8 lines are needed to define and solve the problem (including the instructions SOLVE_PROBLEM and COMPUTE_REACTION) and almost twice as much instructions for getting the required output as needed (mostly PRINTs and one WRITE_RESULTS):

```
PROBLEM mechanical           # self-descriptive
READ_MESH tensile-test.msh   # lengths are in mm

# material properties, E and nu are "special" variables for the mechanical problem
E = 200e3   # [ MPa = N / mm^2 ]
nu = 0.3

# boundary conditions, fixed and Fx are "special" keywords for the mechanical problem
# the names "left" and "right" should match the physical names in the .geo
BC left  fixed
BC right Fx=10e3  # [ N ]

# we can now solve the problem, after this keyword the results will be available for output
```

```
SOLVE_PROBLEM

# essentially we are done by now, we have to write the expected results

# 1. a VTK file to be post-processed in ParaView with
#    a. the displacements [u,v,w] as a vector
#    b. the von Mises stress sigma as a scalar
#    c. the six components of the stress tensor as six scalars
WRITE_MESH tensile-test.vtk VECTOR u v w  sigma sigmax sigmay sigmaz tauxy tauyz tauzx
PRINT "1. post-processing view written in tensile-test.vtk"

# 2. the displacement vector at the center of the specimen
PRINT "2. displacement in x at origin:   " u(0,0,0)   "[ mm ]"
PRINT "   displacement in y at (0,10,0): " v(0,10,0)  "[ mm ]"
PRINT "   displacement in z at (0,0,2.5):" w(0,0,2.5) "[ mm ]"

# 3. the principal stresses at the center
PRINT "3. principal stresses at origin: " %.4f sigma1(0,0,0) sigma2(0,0,0) sigma3(0,0,0) "[  ↩
    MPa ]"

# 4. the reaction at the left surface
COMPUTE_REACTION left  RESULT R_left
PRINT "4. reaction at left surface: " R_left "[ N ]"

# 5. stress concentrations at a sharp edge
PRINT "5. stress concentrations at x=55, y=10, z=2.5 mm"
PRINT "von Mises stress:" sigma(55,10,2.5) "[ MPa ]"
PRINT "Tresca    stress:" sigma1(55,10,2.5)-sigma3(55,10,2.5) "[ MPa ]"
PRINT "stress tensor:"
PRINT %.1f sigmax(55,10,2.5) tauxy(55,10,2.5)  tauzx(55,10,2.5)
PRINT %.1f tauxy(55,10,2.5)  sigmay(55,10,2.5) tauyz(55,10,2.5)
PRINT %.1f tauzx(55,10,2.5)  tauyz(55,10,2.5)  sigmaz(55,10,2.5)
```

Moreover, when solving PDEs, FeenoX will be also smart enough not to compute quantities which are not going to be written anywhere. For example, if the input file does not reference the principal stress `sigma1` (or `WRITE_RESULTS` does not ask for it) then FeenoX will not compute it.

B.3.2.1. Output formats

With the ASCII output to standard output (and other text files) controlled with `PRINT`-like instructions, YAML or JSON outputs can be easily implemented within the input file itself. For example,

```
DEFAULT_ARGUMENT_VALUE 1 "hello world"
phi = (1+sqrt(5))/2

PRINTF "a: %.3f" 1/3
PRINT TEXT "phi:" phi SEP " "
PRINT message: ${1}   SEP " "
```

would give

```
$ feenox yaml.fee | tee test.yaml | yq .
{
  "a": 0.333,
  "phi": 1.61803,
  "message": "hello world"
```

```
}
$ cat test.yaml
a: 0.333
phi: 1.61803
message: hello world
$
```

Now, JSON is more picky and care with quoted characters is needed:

1. Curly brackets `{` and `}` are used for multi-line input in FeenoX so they have to be quoted as `\{` and `\}`.
2. Double quotes `"` are used to delimit keywords with blanks, so they also have to be quoted `\"` when appearing verbatim in an output token.

```
DEFAULT_ARGUMENT_VALUE 1 "hello world"
phi = (1+sqrt(5))/2

PRINTF "\{ \"a\": %.3f," 1/3
PRINT  TEXT "\"phi\":" phi ,
PRINT  "\"message\": \"${1}\" \}"
```

```
$ feenox json.fee | jq .
{
  "a": 0.333,
  "phi": 1.61803,
  "message": "hello world"
}
$
```

In the same sense, in principle any ASCII-based format can be implemented this way. Markdown output, which can then be converted to other formats as well (such as LaTeX which can then create professionally-looking tables as in figure B.25), has been already covered in section B.2.7.

Current version can write space and time-dependent distributions into Gmsh's `.msh` and VTK's `.vtk` formats. Both of them are open standard and have open-source readers. Other formats such as VTK's `.vtu` or CalculiX's `.frd` should be easy to add, but in any case the mesh data converters such as Meshio can be used to convert FeenoX's post-processing output to other formats as well.

B.3.2.2. Data exchange between non-conformal meshes

To illustrate how the output of a FeenoX execution can be read by another FeenoX instance, let us revisit the plane-strain square from section B.3.1.5. This time, instead of setting the temperature with an algebraic expression, we will solve a thermal problem that gives rise to the same temperature distribution but on a different mesh.

First, we solve a thermal problem on the same square $[-1,+1]\times[-1,+1]$ such that the resulting temperature field is $T(x,y)=200+350\cdot y$:

```
PROBLEM thermal 2D
READ_MESH square-centered-unstruct.msh # [-1:+1]x[-1:+1]

BC bottom    T=-150
BC top       T=+550
k = 1

SOLVE_PROBLEM
WRITE_MESH thermal-square-temperature.msh T
```

Now, we read the temperature $T(x, y)$ from the thermal output mesh file `thermal-square-temperature.msh` (which is a triangular unstructured grid) into the mechanical input mesh file `square-centered.msh` (which is a structured quadrangular grid):

```
PROBLEM mechanical plane_strain
READ_MESH square-centered.msh # [-1:+1]x[-1:+1]

# fixed at left, uniform traction in the x direction at right
BC left    fixed
BC right   tx=50

# ASME II Part D pag. 785 Carbon steels with C<=0.30%
FUNCTION E_carbon(temp) INTERPOLATION steffen DATA {
-200  216
-125  212
-75   209
25    202
100   198
150   195
200   192
250   189
300   185
350   179
400   171
450   162
500   151
550   137
}

# read the temperature from a previous result
READ_MESH thermal-square-temperature.msh DIM 2 READ_FUNCTION T

# Young modulus is the function above evaluated at the local temperature
E(x,y) := E_carbon(T(x,y))

# uniform Poisson's ratio
nu = 0.3

SOLVE_PROBLEM
WRITE_MESH mechanical-square-temperature-from-msh.vtk  E T VECTOR u v 0
```

Indeed, the terminal mimic shows the difference between the mechanical input from this section and the one that used an explicit algebraic expression.

```
$ gmsh -2 square-centered-unstruct.geo
[...]
Info    : Done meshing 2D (Wall 0.012013s, CPU 0.033112s)
```

```
Info    : 65 nodes 132 elements
Info    : Writing 'square-centered-unstruct.msh'...
Info    : Done writing 'square-centered-unstruct.msh'
Info    : Stopped on Wed Aug  3 17:47:39 2022 (From start: Wall 0.0208329s, CPU 0.064825s)
$ feenox thermal-square.fee
$ feenox mechanical-square-temperature-from-msh.fee
$ diff mechanical-square-temperature-from-msh.fee mechanical-square-temperature.fee
26,27c26,29
< # read the temperature from a previous result
< READ_MESH thermal-square-temperature.msh DIM 2 READ_FUNCTION T
---
>
> # known temperature distribution
> # (we could have read it from an output of a thermal problem)
> T(x,y) := 200 + 350*y
36c38
< WRITE_MESH mechanical-square-temperature-from-msh.vtk  E T VECTOR u v 0
---
> WRITE_MESH mechanical-square-temperature.vtk  E  VECTOR u v 0
$
```

B.4. Quality assurance

> Since the results obtained with the tool might be used in verifying existing equipment or in designing new mechanical parts in sensitive industries, a certain level of software quality assurance is needed. Not only are best-practices for developing generic software such as
>
> - employment of a version control system,
> - automated testing suites,
> - user-reported bug tracking support.
> - etc.
>
> required, but also since the tool falls in the category of engineering computational software, verification and validation procedures are also mandatory, as discussed below. Design should be such that governance of engineering data including problem definition, results and documentation can be efficiently performed using state-of-the-art methodologies, such as distributed control version systems

The development of FeenoX is tracked with the distributed version control system Git. The official repository is hosted on Github at https://github.com/seamplex/feenox/. New non-trivial features are added in new branches which are then eventually merged into the main branch.

Note that nowadays mentioning that the source code of a piece of software is tracked with Git (why wouldn't it?) is like saying a hotel has a private bathroom in each room (why wouldn't it?). But the reader ought to keep in mind that there is a non-negligible fraction of production calculation codes (even nuclear-related) whose source code is *not* tracked with a DVCS, let alone features and bug fixes follow the branch-review-merge path.

B.4.1. Reproducibility and traceability

> The full source code and the documentation of the tool ought to be maintained under a control version system. Whether access to the repository is public or not is up to the vendor, as long as the copying conditions are compatible with the definitions of both free and open source software from the FSF and the OSI, respectively as required in section A.1.
>
> In order to be able to track results obtained with different version of the tools, there should be a clear release procedure. There should be periodical releases of stable versions that are required
>
> - not to raise any warnings when compiled using modern versions of common compilers (e.g. GNU, Clang, Intel, etc.)
> - not to raise any errors when assessed with dynamic memory analysis tools (e.g. Valgrind) for a wide variety of test cases
> - to pass all the automated test suites as specified in section A.4.2
>
> These stable releases should follow a common versioning scheme, and either the tarballs with the sources and/or the version control system commits should be digitally signed by a cognizant responsible. Other unstable versions with partial and/or limited features might be released either in the form of tarballs or made available in a code repository. The requirement is that unstable tarballs and main (a.k.a. trunk) branches on the repositories have to be compilable. Any feature that does not work as expected or that does not even compile has to be committed into develop branches before being merge into trunk.
>
> If the tool has an executable binary, it should be able to report which version of the code the executable corresponds to. If there is a library callable through an API, there should be a call which returns the version of the code the library corresponds to.
>
> It is recommended not to mix mesh data like nodes and element definition with problem data like material properties and boundary conditions so as to ease governance and tracking of computational models and the results associated with them. All the information needed to solve a particular problem (i.e. meshes, boundary conditions, spatially-distributed material properties, etc.) should be generated from a very simple set of files which ought to be susceptible of being tracked with current state-of-the-art version control systems. In order to comply with this suggestion, ASCII formats should be favored when possible.

As stated in the previous section, the official repository is freely available on Github. As long as the copying conditions (GPLv3+) are met, the repository can be freely cloned and/or forked.

Each binary executable `feenox` has embedded a literal string with the version of the source code used to build it. When running without arguments, it will print the version (which includes the hash of the last commit to the repository) and the usage:

```
$ feenox
FeenoX v1.0.7-g9b98430
a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool

usage: feenox [options] inputfile [replacement arguments] [petsc options]

  -h, --help         display options and detailed explanations of commmand-line usage
  -v, --version      display brief version information and exit
```

```
  -V, --versions     display detailed version information
  -c, --check        validates if the input file is sane or not
  --pdes             list the types of PROBLEMs that FeenoX can solve, one per line
  --elements_info    output a document with information about the supported element types
  --linear           force FeenoX to solve the PDE problem as linear
  --non-linear       force FeenoX to solve the PDE problem as non-linear

Run with --help for further explanations.
$
```

As required by the GNU Standards, running with `-v` or `--version` will print copyright information as well:

```
$ feenox -v
FeenoX v1.0.7-g9b98430
a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Copyright © 2009--2024 Seamplex, https://seamplex.com/feenox
GNU General Public License v3+, https://www.gnu.org/licenses/gpl.html.
FeenoX is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
$
```

And running with `-V` or `--versions` will print detailed versioning information about

1. the date and time of the last commit to the repository
2. the date and time of compilation
3. the architecture, compiler type, version and flags used to build the executable
4. the versions of the external numerical libraries used to link the executable

```
$ feenox --versions
FeenoX v1.0.7-g9b98430
a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Last commit date   : Tue Mar 19 16:17:30 2024 -0300
Build date         : Wed Mar 20 07:40:34 2024 -0300
Build architecture : linux-gnu x86_64
Compiler version   : gcc (Debian 12.2.0-14) 12.2.0
Compiler expansion : gcc -Wl,-z,relro -I/usr/include/x86_64-linux-gnu/mpich -L/usr/lib/x86_64- ↩
    linux-gnu -lmpich
Compiler flags     : -O3 -flto=auto -no-pie
Builder            : [redacted]tom
GSL version        : 2.7.1
SUNDIALS version   : N/A
PETSc version      : Petsc Development GIT revision: v3.20.5-856-g0d3f65ad054  GIT Date:  ↩
    2024-03-20 02:13:21 +0000
PETSc arch         : arch-linux-c-debug
PETSc options      : --download-eigen --download-hdf5 --download-hypre --download-metis -- ↩
    download-mumps --download-parmetis --download-scalapack --download-slepc --with-64-bit- ↩
    indices=no --with-debugging=yes --with-precision=double --with-scalar-type=real PETSC_ARCH= ↩
    arch-linux-c-debug --force
SLEPc version      : SLEPc Development GIT revision: v3.20.1-36-g7a35a7b97  GIT Date: 2023-12-02 ↩
     02:30:03 -0600
$
```

The version is composed of three dot-separated integers:

1. the major version (major changes)
2. the minor version (incompatible input changes)
3. the revision (individual commits from last tag)

The `autogen.sh` script builds this string at compile time, which is stored in a header and finally embedded into the executable. The major m and minor n integers are read from the git tag formatted as `vm.n`, which is bumped manually by adding an annotated tag to a particular commit. The revision is computed automatically with `git describe` as the number of commits in the main branch from the tag to the last commit. The hash is also added to avoid ambiguities in case the repository is forked and diverged from the official one. Periodically, source and binary tarballs are built (using automated scripts in the `dist` subdirectory) and published online.

Given the input-file scheme thoroughfully explained in section B.3.1 the separation of the problem formulation from the mesh data–the input files can be tracked with Git (or any other VCS) as well, therefore enhancing traceability of results and data governance. Again, this might be obvious in the 2020s. But there are many FEM solvers which mix the mesh data with the problem definition (e.g. when external loads have to be given at the nodes instead of using expressions like `p=rho*g*z` or `Fx=1e3`).

B.4.2. Automated testing

> A mean to automatically test the code works as expected is mandatory. A set of problems with known solutions should be solved with the tool after each modification of the code to make sure these changes still give the right answers for the right questions and no regressions are introduced. Unit software testing practices like continuous integration and test coverage are recommended but not mandatory.
>
> The tests contained in the test suite should be
>
> - varied,
> - diverse, and
> - independent
>
> Due to efficiency issues, there can be different sets of tests (e.g. unit and integration tests, quick and thorough tests, etc.) Development versions stored in non-main branches can have temporarily-failing tests, but stable versions have to pass all the test suites.

The `make check` target will execute a set of Bash scripts which will run hundreds of cases and compare their solutions to reference values. These references might be

i. analytical solutions,
ii. known reference solutions, or
iii. random reference solutions.

Depending on the type of case being run, some of these tests might work as very simplified verification cases. But the bulk work as regressions tests so developers adding new features can check they do not break existing working code.

For example, if by mistake a developer flips a sign of one term when setting convection boundary conditions in the heat-conduction PDE, i.e. from

```
double rhs = h*Tref;
```

to

```
double rhs = -h*Tref;
```

then the `make check` step will detect it. In effect,

```
$ make check
[...]
XFAIL: tests/abort.sh
PASS: tests/algebraic_expr.sh
PASS: tests/annulus-modal.sh
PASS: tests/uo2-pellet.sh
[...]
PASS: tests/t21.sh
FAIL: tests/thermal-1d.sh
PASS: tests/thermal-2d.sh
FAIL: tests/thermal-3d.sh
XFAIL: tests/thermal-slab-no-k.sh
XFAIL: tests/thermal-slab-wrong-bc.sh
FAIL: tests/thermal-radiation.sh
PASS: tests/transient-mesh.sh
PASS: tests/trig.sh
[...]
============================================================================
Testsuite summary for feenox 1.0.7
============================================================================
# TOTAL: 75
# PASS:  64
# SKIP:  2
# XFAIL: 6
# FAIL:  3
# XPASS: 0
# ERROR: 0
============================================================================
See ./test-suite.log
Please report to jeremy@seamplex.com
============================================================================
make[3]: *** [Makefile:1723: test-suite.log] Error 1
[...]
make: *** [Makefile:1608: check-recursive] Error 1
$
```

B.4.3. Bug reporting and tracking

A system to allow developers and users to report bugs and errors and to suggest improvements should be provided. If applicable, bug reports should be tracked, addressed and documented. User-provided suggestions might go into the back log or TO-DO list if appropriate.

Here, “bug and errors” mean failure to

- compile on supported architectures,
- run (unexpected run-time errors, segmentation faults, etc.)
- return a correct result

The Github Issues feature at https://github.com/seamplex/feenox/issues is used to report and track bugs and errors (figure B.40).

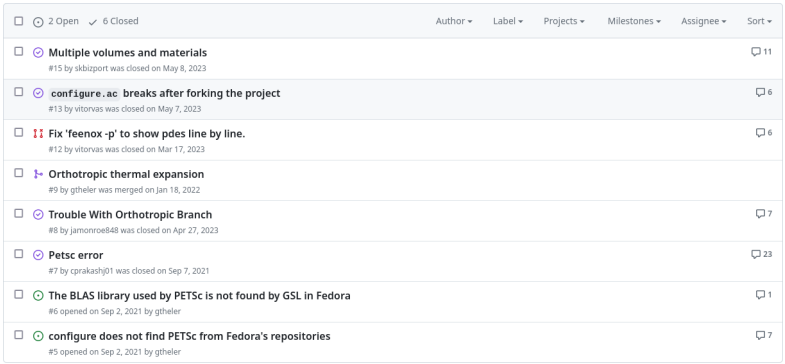


Figure B.40.: Github Issues for FeenoX

B.4.4. Documentation

Documentation should be complete and cover both the user and the developer point of view. It should include a user manual adequate for both reference and tutorial purposes. Other forms of simplified documentation such as quick reference cards or video tutorials are not mandatory but highly recommended. Since the tool should be extendable (section A.2.6), there should be a separate development manual covering the programming design and implementation, explaining how to extend the code and how to add new features. Also, as non-trivial mathematics which should be verified are expected, a thorough explanation of what equations are taken into account and how they are solved is required.

It should be possible to make the full documentation available online in a way that it can be both printed in hard copy and accessed easily from a mobile device. Users modifying the tool to suit their own needs should be able to modify the associated documentation as well, so a clear notice about the licensing terms of the documentation itself (which might be different from the licensing terms of the source code itself) is mandatory. Tracking changes in the documentation should be similar to tracking changes in the code base. Each individual document ought to explicitly state to which version of the tool applies. Plain ASCII formats should be preferred. It is forbidden to submit documentation in a non-free format.

The documentation shall also include procedures for

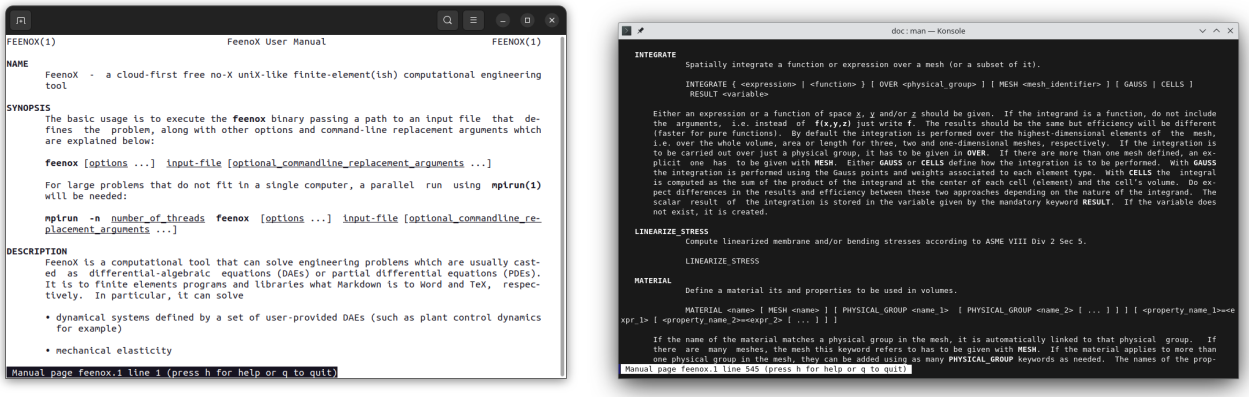
- reporting errors and bugs
- releasing stable versions
- performing verification and validation studies
- contributing to the code base, including
  - code of conduct
  - coding styles
  - variable and function naming conventions

According to Eric Raymond's book "The Art of Unix Programming":

> Compactness is the property that a design can fit inside a human being's head. A good practical test for compactness is this: Does an experienced user normally need a manual? If not, then the design (or at least the subset of it that covers normal use) is compact.

Following to 20-80 rule, we could say that FeenoX is compact for 80% of its usage. But the most complex 20% of the cases might need users (even the author) to look up the syntax of the definition and instructions in the manual page (illustrated in figure B.41), which is accessible with `man feenox` after installing with `make install`:

```
$ man -k feenox
feenox (1)           - a cloud-first free no-X uniX-like finite-element(ish) computational ↩
    engineering tool
$ man feenox
$
```



(a) Gnome Terminal

(b) Konsole

Figure B.41.: The FeenoX Unix manpage in section 1 when running `man feenox`

This man page is compiled into troff from a markdown source, which in turn has some sections involving the syntax and reference of the

- definitions and instructions
- special variables
- internal built-in functions and functionals

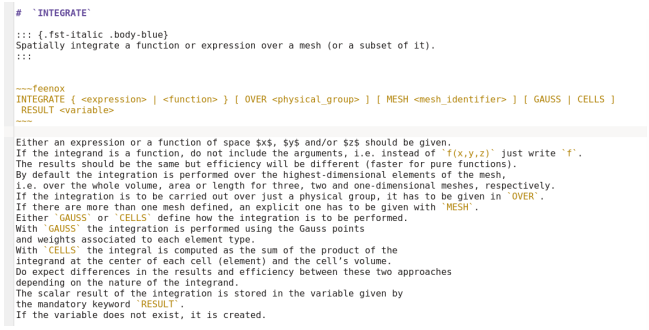
generated by a script that parses the actual source code. For instance, the code that parses the `INTEGRATE` function has three-forward-slash comments that tell this script that it has to prepare documentation:

```
int feenox_parse_integrate(void) {

  mesh_integrate_t *mesh_integrate = NULL;
  feenox_check_alloc(mesh_integrate = calloc(1, sizeof(mesh_integrate_t)));

///kw_pde+INTEGRATE+usage { <expression> | <function> }
///kw_pde+INTEGRATE+detail Either an expression or a function of space $x$, $y$ and/or $z$ should be given.
///kw_pde+INTEGRATE+detail If the integrand is a function, do not include the arguments, i.e. instead of `f(x,y,z)` just write `f`.
///kw_pde+INTEGRATE+detail The results should be the same but efficiency will be different (faster for pure functions).
  char *token = feenox_get_next_token(NULL);
  if ((mesh_integrate->function = feenox_get_function_ptr(token)) == NULL) {
    feenox_call(feenox_expression_parse(&mesh_integrate->expr, token));
  }

  char *name_mesh = NULL;
  char *name_physical_group = NULL;
  char *name_result = NULL;

  while ((token = feenox_get_next_token(NULL)) != NULL) {
///kw_pde+INTEGRATE+usage [ OVER <physical_group> ]
///kw_pde+INTEGRATE+detail By default the integration is performed over the highest-dimensional elements of the mesh,
///kw_pde+INTEGRATE+detail i.e. over the whole volume, area or length for three, two and one-dimensional meshes, respectively.
///kw_pde+INTEGRATE+detail If the integration is to be carried out over just a physical group, it has to be given in `OVER`.

    if (strcasecmp(token, "OVER") == 0) {
      feenox_call(feenox_parser_string(&name_physical_group));

[...]
```

The script `reference.sh` would create the markdown snippet shown in figure B.42a, which then can be converted to other output formats ([[figure figura B.42b;] [figure figura B.42c;] figure B.42d]) for the final user (and author) to look up the syntax of the input keywords.

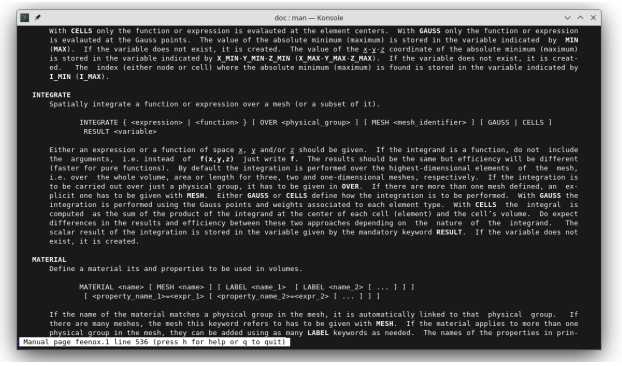
Other pieces of documentation in markdown which then are converted to HTML & PDF (with Pandoc and XeLaTeX) include:

- The FeenoX manual
- The FeenoX description (converted to Texinfo as well)
- Software Requirements Specification
- Software Design Specification
- Frequently Asked Questions
- FeenoX Unix man page
- History
- Compilation guide
- Programming guide



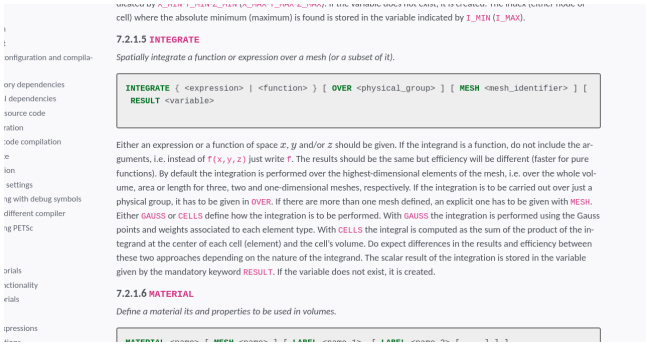
```
#  `INTEGRATE`

::: {.fst-italic .body-blue}
Spatially integrate a function or expression over a mesh (or a subset of it).
:::


~~~feenox
INTEGRATE { <expression> | <function> } [ OVER <physical_group> ] [ MESH <mesh_identifier> ] [ GAUSS | CELLS ]
 RESULT <variable>
~~~

Either an expression or a function of space $x$, $y$ and/or $z$ should be given.
If the integrand is a function, do not include the arguments, i.e. instead of `f(x,y,z)` just write `f`.
The results should be the same but efficiency will be different (faster for pure functions).
By default the integration is performed over the highest-dimensional elements of the mesh,
i.e. over the whole volume, area or length for three, two and one-dimensional meshes, respectively.
If the integration is to be carried out over just a physical group, it has to be given in `OVER`.
If there are more than one mesh defined, an explicit one has to be given with `MESH`.
Either `GAUSS` or `CELLS` define how the integration is to be performed.
With `GAUSS` the integration is performed using the Gauss points
and weights associated to each element type.
With `CELLS` the integral is computed as the sum of the product of the
integrand at the center of each cell (element) and the cell's volume.
Do expect differences in the results and efficiency between these two approaches
depending on the nature of the integrand.
The scalar result of the integration is stored in the variable given by
the mandatory keyword `RESULT`.
If the variable does not exist, it is created.
```

(a) Markdown



```
doc : man — Konsole
With CELLS only the function or expression is evaluated at the element centers.  With GAUSS only the function or expression
is evaluated at the Gauss points.  The value of the absolute minimum (maximum) is stored in the variable indicated by  MIN
(MAX).  If the variable does not exist, it is created.  The value of the x-y-z coordinate of the absolute minimum (maximum)
is stored in the variable indicated by X_MIN-Y_MIN-Z_MIN (X_MAX-Y_MAX-Z_MAX).  If the variable does not exist, it is creat-
ed.  The index (either node or cell) where the absolute minimum (maximum) is found is stored in the variable indicated by
I_MIN (I_MAX).

INTEGRATE
    Spatially integrate a function or expression over a mesh (or a subset of it).

          INTEGRATE { <expression> | <function> } [ OVER <physical_group> ] [ MESH <mesh_identifier> ] [ GAUSS | CELLS ]
           RESULT <variable>

    Either an expression or a function of space x, y and/or z should be given.  If the integrand is a function, do not  include
    the  arguments,  i.e. instead  of  f(x,y,z)  just write f.  The results should be the same but efficiency will be different
    (faster for pure functions).  By default the integration is performed over the highest-dimensional elements  of  the  mesh,
    i.e. over  the whole volume, area or length for three, two and one-dimensional meshes, respectively.  If the integration is
    to be carried out over just a physical group, it has to be given in OVER.  If there are more than one mesh defined, an  ex-
    plicit one has to be given with MESH.  Either GAUSS or CELLS define how the integration is to be performed.  With GAUSS the
    integration is performed using the Gauss points and weights associated to each element type.  With CELLS  the  integral  is
    computed  as the sum of the product of the integrand at the center of each cell (element) and the cell's volume.  Do expect
    differences in the results and efficiency between these two approaches depending on  the  nature  of  the  integrand.  The
    scalar result of the integration is stored in the variable given by the mandatory keyword RESULT.  If the variable does not
    exist, it is created.

MATERIAL
    Define a material its and properties to be used in volumes.

          MATERIAL <name> [ MESH <name> ] [ LABEL <name_1>  [ LABEL <name_2> [ ... ] ] ]
           [ <property_name_1>=<expr_1> [ <property_name_2>=<expr_2> [ ... ] ] ]

    If the name of the material matches a physical group in the mesh, it is automatically linked to that  physical  group.  If
    there are many meshes, the mesh this keyword refers to has to be given with MESH.  If the material applies to more than one
    physical group in the mesh, they can be added using as many LABEL keywords as needed.  The names of the properties in prin-
Manual page feenox.1 line 536 (press h for help or q to quit)
```

(b) Manpage



cell) where the absolute minimum (maximum) is found is stored in the variable indicated by I_MIN (I_MAX).

7.2.1.5 INTEGRATE

Spatially integrate a function or expression over a mesh (or a subset of it).

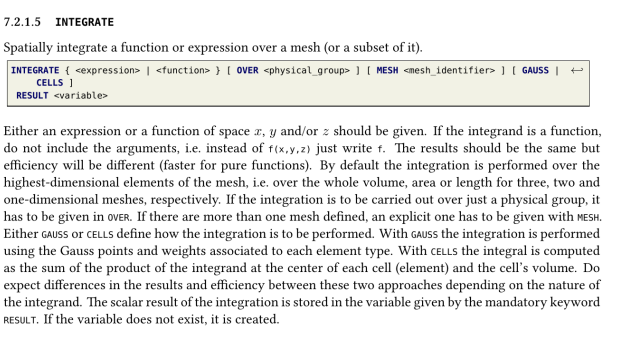
```
INTEGRATE { <expression> | <function> } [ OVER <physical_group> ] [ MESH <mesh_identifier> ] [
 RESULT <variable>
```

Either an expression or a function of space x, y and/or z should be given. If the integrand is a function, do not include the arguments, i.e. instead of `f(x,y,z)` just write `f`. The results should be the same but efficiency will be different (faster for pure functions). By default the integration is performed over the highest-dimensional elements of the mesh, i.e. over the whole volume, area or length for three, two and one-dimensional meshes, respectively. If the integration is to be carried out over just a physical group, it has to be given in OVER. If there are more than one mesh defined, an explicit one has to be given with MESH. Either GAUSS or CELLS define how the integration is to be performed. With GAUSS the integration is performed using the Gauss points and weights associated to each element type. With CELLS the integral is computed as the sum of the product of the integrand at the center of each cell (element) and the cell's volume. Do expect differences in the results and efficiency between these two approaches depending on the nature of the integrand. The scalar result of the integration is stored in the variable given by the mandatory keyword RESULT. If the variable does not exist, it is created.

7.2.1.6 MATERIAL

Define a material its and properties to be used in volumes.

(c) HTML



7.2.1.5 INTEGRATE

Spatially integrate a function or expression over a mesh (or a subset of it).

```
INTEGRATE { <expression> | <function> } [ OVER <physical_group> ] [ MESH <mesh_identifier> ] [ GAUSS |
    CELLS ]
 RESULT <variable>
```

Either an expression or a function of space x, y and/or z should be given. If the integrand is a function, do not include the arguments, i.e. instead of `f(x,y,z)` just write `f`. The results should be the same but efficiency will be different (faster for pure functions). By default the integration is performed over the highest-dimensional elements of the mesh, i.e. over the whole volume, area or length for three, two and one-dimensional meshes, respectively. If the integration is to be carried out over just a physical group, it has to be given in OVER. If there are more than one mesh defined, an explicit one has to be given with MESH. Either GAUSS or CELLS define how the integration is to be performed. With GAUSS the integration is performed using the Gauss points and weights associated to each element type. With CELLS the integral is computed as the sum of the product of the integrand at the center of each cell (element) and the cell's volume. Do expect differences in the results and efficiency between these two approaches depending on the nature of the integrand. The scalar result of the integration is stored in the variable given by the mandatory keyword RESULT. If the variable does not exist, it is created.

(d) PDF

Figure B.42.: Reference for the keyword INTEGRATE in Markdown created out of special comments in the C source converted to different output formats.

C. FeenoX and the rules of Unix philosophy

> The Unix philosophy is not a formal design method. It wasn't handed down from the high fastnesses of theoretical computer science as a way to produce theoretically perfect software. Nor is it that perennial executive's mirage, some way to magically extract innovative but reliable software on too short a deadline from unmotivated, badly managed, and underpaid programmers.
> The Unix philosophy (like successful folk traditions in other engineering disciplines) is bottom-up, not top-down. It is pragmatic and grounded in experience. It is not to be found in official methods and standards, but rather in the implicit half-reflexive knowledge, the expertise that the Unix culture transmits. It encourages a sense of proportion and skepticism—and shows both by having a sense of (often subversive) humor.
> *Eric Raymond*, *The Art of UNIX Programming*, *2003*

In 1978, Doug McIlroy—the inventor of Unix pipes and one of the founders of the Unix tradition—stated:

i. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new features.

ii. Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.

iii. Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.

iv. Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.

He later summarized it this way:

> This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

FeenoX explicitly followed the above ideas from scratch, especially the for sentences in bullet ii. It is even, like Unix itself, a third-system effect where clumsy parts of previous attempts were thrown away and rebuilt from scratch. The following sections explain how each of the seventeen rules was taken into account when designing and implementing FeenoX.

C.1. Rule of Modularity

> Developers should build a program out of simple parts connected by well defined interfaces, so problems are local, and parts of the program can be replaced in future versions to support new features. This rule aims to save time on debugging code that is complex, long, and unreadable.

FeenoX is designed to be as lightweight as possible. On the one hand, it relies on third-party high-quality libraries to do the heavy mathematical weightlifting such as

- GNU Scientific Library for general mathematics,
- SUNDIALS IDA for ODEs and DAEs,
- PETSc for linear, non-linear and transient PDEs, and
- SLEPc for PDEs involving eigen problems

because these libraries were written by professional programmers using algorithms designed by professional mathematicians. Yet-to-be-discovered improved mathematical schemes and/or coding algorithms can be eventually used by FeenoX by just updating those dependencies, which for sure will keep their well-defined interfaces (because they are programmed by professional programmers).

Moreover, the extensibility feature (section C.17) of having each PDE in separate directories which can be added or removed at compile time without changing any line of the source code goes into this direction as well. Relying of C function pointers allows (in principle) to replace these "virtual" methods with other ones using the same interface.

> Note that our (human) languages in general and words in particular shape and limit the way we think. Fortran's concept of "modules" is *not* the same as Unix's concept of "modularity." I wish two different words had been used.

C.2. Rule of Clarity

> Developers should write programs as if the most important communication is to the developer who will read and maintain the program, rather than the computer. This rule aims to make code as readable and comprehensible as possible for whoever works on the code in the future.

Of course there might be a confirmation bias in this section because every programmer thinks their code is clear (and everybody else's is not). But the first design decision to fulfill this rule is the programming language: there is little change to fulfill it with Fortran. One might argue that C++ can be clearer than C in some points, but for the vast majority of the source code they are equally clear. Besides, C is far simpler than C++ (see rule of simplicity).

The second decision is not about the FeenoX source code but about FeenoX inputs: clear human-readable input files without any extra unneeded computer-level nonsense. The two illustrative cases are the NAFEMS LE10 & LE11 benchmarks, where there is a clear one-to-one correspondence between the "engineering" formulation and the input file FeenoX understands.

C.3. Rule of Composition

> Developers should write programs that can communicate easily with other programs. This rule aims to allow developers to break down projects into small, simple programs rather than overly complex monolithic programs.

Previous designs of FeenoX' predecessors used to include instructions to perform parametric sweeps(and even optimization loops), non-trivial macro expansions using M4 and even execution of arbitrary shell commands. These non-trivial operations were removed from FeenoX to focus on the rule of composition, paying especially attention to easing the inclusion of calling the `feenox` binary from shell scripts, enforcing the composition with other Unix-like tools. Emphasis has been put on adding flexibility to programmatic generation of input files (see also rule of generation in section C.14) and the handling and expansion of command-line arguments to increase the composition with other programs.

Moreover, the output is 100% controlled by the user at run-time so it can be tailored to suit any other programs' input needs as well. An illustrative example is creating professional-looking tables with results using AWK & LaTeX.

C.4. Rule of Separation

> Developers should separate the mechanisms of the programs from the policies of the programs; one method is to divide a program into a front-end interface and a back-end engine with which that interface communicates. This rule aims to prevent bug introduction by allowing policies to be changed with minimum likelihood of destabilizing operational mechanisms.

FeenoX relies of the rule of separation (which also links to the next two rules of simplicity and parsimony) from the very beginning of its design phase. It was explicitly designed as a glue layer between a mesher like Gmsh and a post-processor like Gnuplot, Gmsh or Paraview. This way, not only flexibility and diversity (see #sec-unix-diversity) can be boosted, but also technological changes can be embraced with little or no effort. For example, CAEplex provides a web-based platform for performing thermo-mechanical analysis on the cloud running from the browser. Had FeenoX been designed as a traditional desktop-GUI program, this would have been impossible. If in the future

CAD/CAE interfaces migrate into virtual and/or augmented reality with interactive 3D holographic input/output devices, the development effort needed to use FeenoX as the back end is negligible.

C.5. Rule of Simplicity

> Developers should design for simplicity by looking for ways to break up program systems into small, straightforward cooperating pieces. This rule aims to discourage developers' affection for writing "intricate and beautiful complexities" that are in reality bug prone programs.

The main source of simplicity comes from the design of the syntax of the input files, discussed in detail in the SDS:

- English-like self-evident input files matching as close as possible the problem text.
- Simple problems need simple input.
- Similar problems need similar inputs.
- If there is a single material there is no need to link volumes to properties.

C.6. Rule of Parsimony

> Developers should avoid writing big programs. This rule aims to prevent overinvestment of development time in failed or suboptimal approaches caused by the owners of the program's reluctance to throw away visibly large pieces of work. Smaller programs are not only easier to write, optimize, and maintain; they are easier to delete when deprecated.

We already said that FeenoX is a glue layer between a mesher and a post-processing tool. Even more, at another level, it acts as two glue layers between the mesher and PETSc, and PETSc and the post-processor.

On the other hand, we also already stated that FeenoX was written from scratch after throwing away clumsy code from two previous attempts. For instance, these previous versions used to implement parametric and optimization schemes. Instead, in FeenoX, these type of runs have to be driven from an outer script (Bash, Python, etc.)

C.7. Rule of Transparency

> Developers should design for visibility and discoverability by writing in a way that their thought process can lucidly be seen by future developers working on the project and using input and output formats that make it easy to identify valid input and correct output. This rule aims to reduce debugging time and extend the lifespan of programs.

As with the rule of clarity (section C.2), there is a risk of falling into the confirmation bias because every programmer thinks its code is transparent. Anyway, FeenoX is written in C99 which is way easier to debug than both Fortran and C++. Yet, very much like PETSc, FeenoX makes use of structures and function pointers to give the same functionality as C++'s virtual methods without needing to introduce other complexities that make the code base harder to maintain and to debug.

Regarding identification of valid inputs and correct outputs,

1. The build system includes a `make check` target that runs hundreds of regressions tests.
2. The code supports verification using the Method of Manufactured Solutions

C.8. Rule of Robustness

> Developers should design robust programs by designing for transparency and discoverability, because code that is easy to understand is easier to stress test for unexpected conditions that may not be foreseeable in complex programs. This rule aims to help developers build robust, reliable products.

Robustness is the child of transparency and simplicity.

C.9. Rule of Representation

> Developers should choose to make data more complicated rather than the procedural logic of the program when faced with the choice, because it is easier for humans to understand complex data compared with complex logic. This rule aims to make programs more readable for any developer working on the project, which allows the program to be maintained.

There is a trade off between clarity and efficiency. However, avoiding Fortran should already fulfill this rule. FeenoX uses C structures with function pointers, which make it far simple to understand than similar Fortran-based FEM tools. Just compare the source directories of FeenoX and CalculiX. Take for instance the file `stress.c` from `src/pdes/mechanical` (which if deleted, will remove support for `mechanical` problems but it will not prevent the compilation of `feenox`) from the former and `calcstress.f` (buried inside 2,400 files in `src`) from the latter. There might be more illustrative examples showing how FeenoX' design is more representative than of CalculiX, but it is way too hard to understand the source code of the latter (even though the license is supposed to be GPL).

C.10. Rule of Least Surprise

> Developers should design programs that build on top of the potential users' expected knowledge; for example, '+' in a calculator program should always mean 'addition'. This rule aims to encourage developers to build intuitive products that are easy to use.

The rules of input syntax have been designed with this rule in mind. Just note a couple of them:

- The command-line arguments after the input file are available to be expanded verbatim in the input file as `$1`, `$2`, etc. (or `${1}`, `${2}`, etc. if they appear in the middle of strings). This syntax matches Bash' syntax for expanding command-line arguments, so any person reading an input file with this syntax already knows what it does. ´
- If one needs a problem where the conductivity depends on x as $k(x) = 1 + x$ then the input is

```
k(x) = 1+x
```

- If a problem needs a temperature distribution given by an algebraic expression $T(x, y, z) = \sqrt{x^2 + y^2} + z$ then do

```
T(x,y,z) = sqrt(x^2+y^2) + z
```

- This syntax for (basic) algebraic expressions matches the common syntax found in Gmsh, Maxima and many other scientific tools. More complex expressions (e.g. involving hyperbolic tangents) might differ slightly.

C.11. Rule of Silence

> Developers should design programs so that they do not print unnecessary output. This rule aims to allow other programs and developers to pick out the information they need from a program's output without having to parse verbosity.

TL;DR: no `PRINT` (or `WRITE_RESULTS`), no output.

C.12. Rule of Repair

> Developers should design programs that fail in a manner that is easy to localize and diagnose or in other words "fail noisily". This rule aims to prevent incorrect output from a program from becoming an input and corrupting the output of other code undetected.

Input errors are detected before the computation is started:

```
$ feenox thermal-error.fee
error: undefined thermal conductivity 'k'
$
```

Run-time errors (even inside the numerical libraries) are caught with custom handlers.

C.13. Rule of Economy

> Developers should value developer time over machine time, because machine cycles today are relatively inexpensive compared to prices in the 1970s. This rule aims to reduce development costs of projects.

As explained in the SDS, output is 100% user-defined so only the desired results are directly obtained instead of needing further digging into tons of undesired data. The approach of "compute and write everything you can in one single run" made sense in 1970 where CPU time was more expensive than human time, but not anymore. Once again, the iconic examples are the NAFEMS LE10 & LE11 benchmarks, where just the required scalar stress at the required location is written into the standard output.

C.14. Rule of Generation

> Developers should avoid writing code by hand and instead write abstract high-level programs that generate code. This rule aims to reduce human errors and save time.

Some key points:

- Input files are M4-like-macro friendly.
- Parametric runs can be done from scripts through expansion of command line arguments.
- Documentation is created out of simple Markdown sources and assembled as needed.

More saliently, the automatic detection of the available PDEs in `src/pdes` is an example of this rule. The `autogen.sh` would loop over each subdirectory and create a source file `src/pdes/parser.c` with a function `feenox_pde_parse_problem_type()` which then will be part of the actual FeenoX source base as the entry point for parsing the `PROBLEM` keyword.

C.15. Rule of Optimization

> Developers should prototype software before polishing it. This rule aims to prevent developers from spending too much time for marginal gains.

FeenoX is still "premature" for heavy optimization. Yet, it is (relatively) faster than other alternatives. It does use link-time optimization to allow for inlining of small routines. There is even a FeenoX benchmarking repository that uses Google's Benchmark library to prototype code optimization: https://github.com/seamplex/feenox-benchmark.

C.16. Rule of Diversity

> Developers should design their programs to be flexible and open. This rule aims to make programs flexible, allowing them to be used in ways other than those their developers intended.

FeenoX can read Gmsh files, but they need not necessarily be created by Gmsh. Other meshing formats (VTK with group names?) are planned to be implemented. Also, either Gmsh or Paraview can be used to post-process results. But also other formats are planned. See section C.17. Diversity is embraced from the bottom up!

C.17. Rule of Extensibility

> Developers should design for the future by making their protocols extensible, allowing for easy plugins without modification to the program's architecture by other developers, noting the version of the program, and more. This rule aims to extend the lifespan and enhance the utility of the code the developer writes.

The main extensibility feature is that each PDE has a separate source directory. Any of them can be used as as template to add new PDEs, which are detected at compile time by the Autotools bootstrapping script.

A final note is that FeenoX is GPLv3+. First, this means that extensions and contributions are welcome. Each author retains the copyright on the contributed code (as long as it is free software). Second, the + is there for the future.

D. FeenoX history

Very much like Unix in the late 1960s and C in the early 1970s, FeenoX is a third-system effect: I wrote a first hack that seemed to work better than I had expected. Then I tried to add a lot of features and complexities which I felt the code needed. After ten years of actual usage, I then realized

1. what was worth keeping,
2. what needed to be rewritten and
3. what had to be discarded.

The first version was called wasora, the second was "The wasora suite" (i.e. a generic framework plus a bunch of "plugins", including a thermo-mechanical one named Fino) and then finally FeenoX. The story that follows explains why I wrote the first hack to begin with.

It was at the movies when I first heard about dynamical systems, non-linear equations and chaos theory. The year was 1993, I was ten years old and the movie was Jurassic Park. Dr. Ian Malcolm (the character portrayed by Jeff Goldblum) explained sensitivity to initial conditions in a memorable scene, which is worth watching again and again. Since then, the fact that tiny variations may lead to unexpected results has always fascinated me. During high school I attended a very interesting course on fractals and chaos that made me think further about complexity and its mathematical description. Nevertheless, it was not not until college that I was able to really model and solve the differential equations that give rise to chaotic behavior.

Figura D.1.: Dr. Ian Malcolm (Jeff Goldblum) explains sensitivity to initial conditions.

In fact, initial-value ordinary differential equations arise in a great variety of subjects in science and engineering. Classical mechanics, chemical kinetics, structural dynamics, heat transfer analysis and dynamical systems, among other disciplines, heavily rely on equations of the form

$$\dot{\mathbf{x}} = F(\mathbf{x}, t)$$

During my years of undergraduate student (circa 2004–2007), whenever I had to solve these kind of equations I had to choose one of the following three options:

1. to program an *ad-hoc* numerical method such as Euler or Runge-Kutta, matching the requirements of the system of equations to solve, or
2. to use a standard numerical library such as the GNU Scientific Library and code the equations to solve into a C program (or maybe in Python), or
3. to use a high-level system such as Octave, Maxima, or some non-free (and worse, see below) programs.

Of course, each option had its pros and its cons. But none provided the combination of advantages I was looking for, namely flexibility (option one), efficiency (option two) and reduced input work (partially given by option three). Back in those days I ended up wandering between options one and two, depending on the type of problem I had to solve. However, even though one can, with some effort, make the code read some parameters from a text file, any other drastic change usually requires a modification in the source code—some times involving a substantial amount of work—and a further recompilation of the code. This was what I most disliked about this way of working, but I could nevertheless live with it.

Regardless of this situation, during my last year of Nuclear Engineering, the tipping point came along. Here's a slightly-fictionalized of a dialog between myself and the teacher at the computer lab (Dr E.), as it might have happened (or not):

> — (Prof.) Open MATLAB.™
> — (Me) It's not installed here. I type `mathlab` and it does not work.
> — (Prof.) It's spelled `matlab`.
> — (Me) Ok, working. (A screen with blocks and lines connecting them appears)
> — (Me) What's this?
> — (Prof.) The point reactor equations.
> — (Me) It's not. These are the point reactor equations:
>
> $$\begin{cases} \dot{\phi}(t) = \dfrac{\rho(t) - \beta}{\Lambda} \cdot \phi(t) + \displaystyle\sum_{i=1}^{N} \lambda_i \cdot c_i \\ \dot{c}_i(t) = \dfrac{\beta_i}{\Lambda} \cdot \phi(t) - \lambda_i \cdot c_i \end{cases}$$
>
> — (Me) And in any case, I'd write them like this in a computer:
>
> ```
> phi_dot = (rho-Beta)/Lambda * phi + sum(lambda[i], c[i], i, 1, N)
> c_dot[i] = beta[i]/Lambda * phi - lambda[i]*c[i]
> ```

This conversation forced me to re-think the ODE-solving issue. I could not (and still cannot) understand why somebody would prefer to solve a very simple set of differential equations by drawing blocks and connecting them with a mouse with no mathematical sense whatsoever. Fast forward fifteen years, and what I wrote above is essentially how one would solve the point kinetics equations with FeenoX.

E. Frequently Asked Questions

E.1. What is FeenoX?

It is "cloud-first a free no-fee no-X uniX-like finite-element(ish) computational engineering tool." Essentially, a finite-element program with a particular design basis:

> FeenoX is to finite-element programs and libraries what Markdown is to word processors (like Word) and typesetting systems (like TeX), respectively.

In increasing order of complexity and comprehensiveness, these resources explain what FeenoX is:

- The examples will give a brief overview of what FeenoX can do.
- The tutorials will walk your through how to use FeenoX to solve problems.
- The README in the GitHub repository has a brief introduction (after explaining why).
- [redacted](2024). FeenoX: a cloud-first finite-element(ish) computational engineering tool. Journal of Open Source Software, 9(95), 5846. https://doi.org/10.21105/joss.05846
- There is also a description in the documentation.
- FeenoX is an "offer" to a fictitious "tender" for a computational tool. The RFQ is the Software Requirements Specification and the explanation of how FeenoX addresses each requirement is the Software Design Specification.
- This presentation from August 2021 explains the SRS/SDS pair. The sources and the examples can be found in this Github repository. There is a recording of the presentation (audio is in Spanish).
- Finally the manual will be the ultimate guide.

E.2. How should I cite FeenoX?

If you use FeenoX and need to cite it, use this BiBTeX entry that points to the 2024 paper in JOSS:

```
@article{feenox-2024,
author = [redacted]
doi = {10.21105/joss.05846},
journal = {Journal of Open Source Software},
month = mar,
number = {95},
pages = {5846},
title = {{FeenoX: a cloud-first finite-element(ish) computational engineering tool}},
url = {https://joss.theoj.org/papers/10.21105/joss.05846},
volume = {9},
year = {2024}
}
```

If you are not using BiBTeX (which you should), just use the plain-text APA format:

[redacted](2024). FeenoX: a cloud-first finite-element(ish) computational engineering tool. Journal of Open Source Software, 9(95), 5846. https://doi.org/10.21105/joss.05846

E.3. What does FeenoX mean?

It does not mean anything particular, but

- The last X makes it rhyme with Unix and Linux.
- "noX" means that there is no graphical (i.e. *X*) interface
- Fee-no means that there are no fees involved (free as in "free beer")
- FeenoX is the successor of the now-superseded FEA program Fino
- It rhymes with FEniCS
- With some luck one can read "Finite ElEments NO-X"
- With mode luck, "FrEE" (as in "free speech")

E.4. How should FeenoX be pronounced?

It would be something like *fee-naaks*: /fiːnɒks/
But whatever works for you is fine.

E.5. Why nothing happens when I double click on `feenox.exe`?

Because by design, FeenoX does not have a Graphical User Interface. Instead, it works like a transfer function between one or more input files and zero or more output files:

```
                           +------------+
mesh (*.msh)  }            |            |             { terminal
data (*.dat)  } input ---> |   FeenoX   |----> output { data files
input (*.fee) }            |            |             { post (vtk/msh)
                           +------------+
```

Recall that FeenoX is designed as a cloud-first tool, and "the cloud" runs on Unix (essentially GNU/Linux) so FeenoX is based on the Unix philosophy. In this world, programs act like filters (i.e. transfer functions). They are executed from a command-line terminal. So instead of "double clicking" the executable, one has to open a terminal and execute it there. Without any arguments it says how to use it:

```
> feenox.exe
FeenoX v0.2.14-gbbf48c9
a free no-fee no-X uniX-like finite-element(ish) computational engineering tool

usage: feenox [options] inputfile [replacement arguments] [petsc options]

  -h, --help         display options and detailed explanations of commmand-line usage
  -v, --version      display brief version information and exit
  -V, --versions     display detailed version information

Run with --help for further explanations.
>
```

With `-h` it gives more information:

```
> feenox.exe -h
usage: feenox [options] inputfile [replacement arguments] [petsc options]

  -h, --help         display options and detailed explanations of commmand-line usage
  -v, --version      display brief version information and exit
  -V, --versions     display detailed version information

  --progress         print ASCII progress bars when solving PDEs
  --mumps            ask PETSc to use the direct linear solver MUMPS
  --linear           force FeenoX to solve the PDE problem as linear
  --non-linear       force FeenoX to solve the PDE problem as non-linear

Instructions will be read from standard input if ""- is passed as
inputfile, i.e.

    $ echo 'PRINT 2+2' | feenox -
    4

The optional [replacement arguments] part of the command line mean that
each argument after the input file that does not start with an hyphen
will be expanded verbatim in the input file in each occurrence of $1,
$2, etc. For example

    $ echo 'PRINT $1+$2' | feenox - 3 4
    7

PETSc and SLEPc options can be passed in [petsc options] as well, with
the difference that two hyphens have to be used instead of only once.
For example, to pass the PETSc option -ksp_view the actual FeenoX
invocation should be

    $ feenox input.fee --ksp_view

See https://www.seamplex.com/feenox/examples for annotated examples.
```

```
Report bugs at https://github.com/seamplex/feenox/issues
Ask questions at https://github.com/seamplex/feenox/discussions
Feenox home page: https://www.seamplex.com/feenox/
>
```

It is explained there that the main input file has to be given as the first argument. So go to the tests or examples directory, find a test you like and run it:

```
> cd tests
> feenox parallelepiped.fee
0.000295443
>
```

In any case, recall once again that FeenoX is a cloud-first tool, and Windows is not cloud-friendly, let alone cloud-first. It is time to re-think what you expect from a finite-element(ish) tool. If you still need a GUI, please check CAEplex.

Try to avoid Windows as much as you can. The binaries are provided as transitional packages for people that for some reason still use such an outdated, anachronous, awful and invasive operating system. They are compiled with Cygwin and have no support whatsoever. Really, really, **get rid of Windows ASAP**.

> "It is really worth any amount of time and effort to get away from Windows if you are doing computational science."
>
> https://lists.mcs.anl.gov/pipermail/petsc-users/2015-July/026388.html

E.6. How do I create input decks for FeenoX?

FeenoX does not have "input decks." It has "input files," which are syntactically-sugared English-like plain-text ASCII files that describe the problem to be solved. First see the examples and the test directory. Then read the documentation.

There are syntax highlighting files for Kate and for Vim that helps the edition of input files. Contributions for other editors (emacs?) are welcome.

E.7. Does FeenoX support beam and/or shell elements?

No, it currently supports solid elements only. Therefore, three-dimensional problems need to have tetrahedra, hexahedra, prisms and/or pyramids; and two-dimensional problems need to have triangles or quadrangles.

It might support non-solid elements for elasticity in future versions, though. Contributions are welcome. Check out the contributing guidelines.

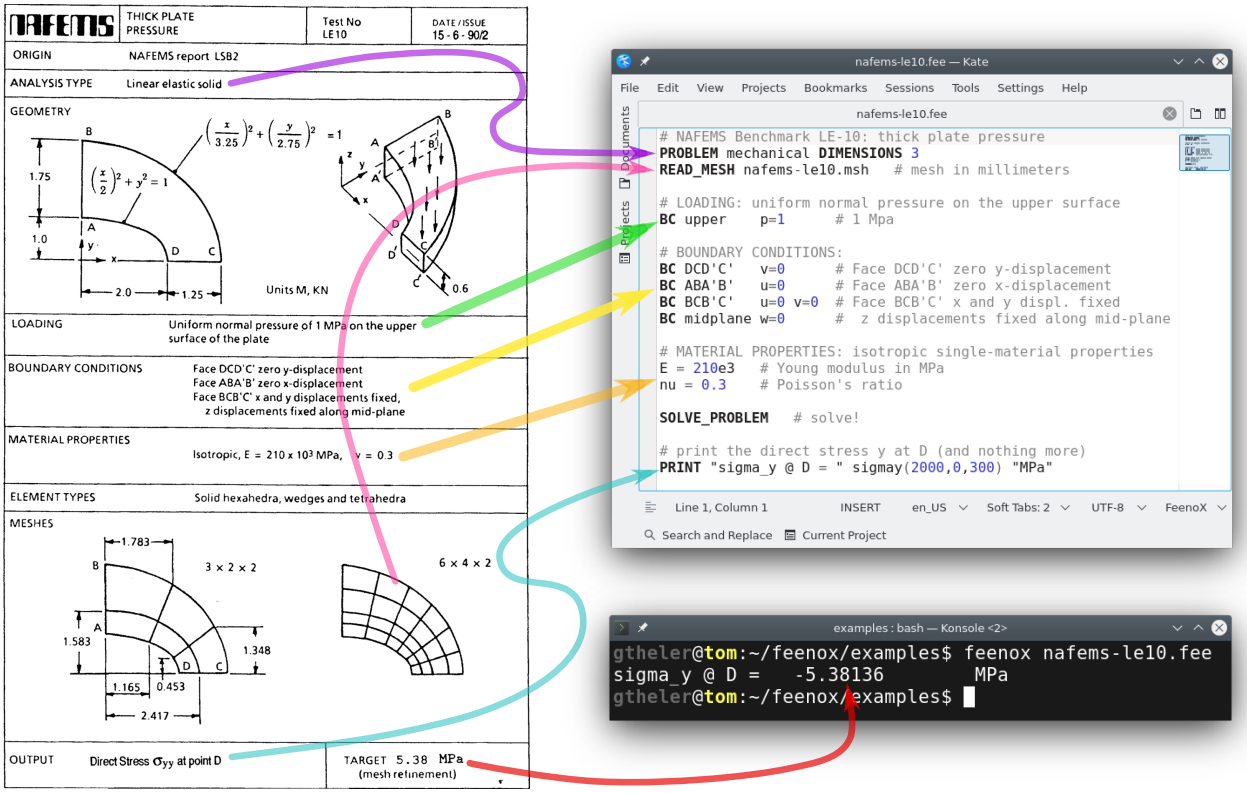


Figura E.1.: The Kate Text Editor can be used to prepare input files with syntax highlighting.

E.8. What license does FeenoX have?

TL;DR:

- The code is GPLv3+: you can use it, modify it and re-distribute it freely (as in free speech) as long as you keep the same licensing terms.
- The documentation is released under the terms of the GNU Free Documentation License version 1.3 or, at your option, any later version: same thing but with particular considerations for documentation instead of code.

FeenoX is licensed under the terms of the GNU General Public License version 3 or, at the user convenience, any later version. This means that users get the four essential freedoms:[1]

0. The freedom to *run* the program as they wish, for *any* purpose.
1. The freedom to *study* how the program works, and *change* it so it does their computing as they wish.
2. The freedom to *redistribute* copies so they can help others.

[1] There are some examples of pieces of computational software which are described as "open source" in which even the first of the four freedoms is denied. The most iconic case is that of Android, whose sources are readily available online but there is no straightforward way of updating one's mobile phone firmware with a customized version, not to mention vendor and hardware lock ins and the possibility of bricking devices if something unexpected happens. In the nuclear industry, it is the case of a Monte Carlo particle-transport program that requests users to sign an agreement about the objective of its usage before allowing its execution. The software itself might be open source because the source code is provided after signing the agreement, but it is not free (as in freedom) at all.

3. The freedom to *distribute* copies of their *modified* versions to others.

So a free program has to be open source, but it also has to explicitly provide the four freedoms above both through the written license and through appropriate mechanisms to get, modify, compile, run and document these modifications using well-established and/or reasonable straightforward procedures. That is why licensing FeenoX as GPLv3+ also implies that the source code and all the scripts and makefiles needed to compile and run it are available for anyone that requires it (i.e. it is compiled with `./configure && make`). Anyone wanting to modify the program either to fix bugs, improve it or add new features is free to do so. And if they do not know how to program, the have the freedom to hire a programmer to do it without needing to ask permission to the original authors. Even more, the documentation is released under the terms of the GNU Free Documentation License so these new (or modified) features can be properly documented as well.

Nevertheless, since these original authors are the copyright holders, they still can use it to either enforce or prevent further actions from the users that receive FeenoX under the GPLv3+. In particular, the license allows re-distribution of modified versions only if

a. they are clearly marked as different from the original, and
b. they are distributed under the same terms of the GPLv3+.

There are also some other subtle technicalities that need not be discussed here such as

- what constitutes a modified version (which cannot be redistributed under a different license)
- what is an aggregate (in which each part be distributed under different licenses)
- usage over a network and the possibility of using AGPL instead of GPL to further enforce freedom

These issues are already taken into account in the FeenoX licensing scheme.

It should be noted that not only is FeenoX free and open source, but also all of the libraries it depends on (and their dependencies) also are. It can also be compiled using free and open source build tool chains running over free and open source operating systems.

E.9. Why is FeenoX written in C and not in...

See the programming guide for further discussion.

E.9.1. C++?

Let us first start with some generalities

Why is C still in use even though we have C++? https://www.quora.com/Why-is-C-still-in-use-even-though-we-have-C++-Is-there-anything-that-C-can-do-but-C++-cant-or-maybe-something-that-is-easier-to-do-in-C-rather-than-C++

As a C programmer, why didn't you switch to C++ in your career? https://qr.ae/pGzfAO

Why is PETSc programmed in C, instead of Fortran or C++? C enables us to build data structures for storing sparse matrices, solver information, etc. in ways that Fortran simply does not allow. ANSI C is a complete standard that all modern C compilers support. The language is identical on all machines. C++ is still evolving and compilers on different machines are not identical. Using C function pointers to provide data encapsulation and polymorphism allows us to get many of the advantages of C++ without using such a large and more complicated language. It would be natural and reasonable to have coded PETSc in C++; we opted to use C instead.

https://www.mcs.anl.gov/petsc/documentation/faq.html#why-c

Why Git is written in C and not in C++, by Linus Torvalds C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it, to the point where it's much much easier to generate total and utter crap with it. Quite frankly, even if the choice of C were to do *nothing* but keep the C++ programmers out, that in itself would be a huge reason to use C.

http://harmful.cat-v.org/software/c++/linus

In particular, I think that even though object-oriented programming does provide a heck of a paradigm (hey, I actually rewrote my Blackjack engine in C++ from its first version in C), it might subtly "force" people to do stuff that is either way too

- convoluted
- artificial
- hard to debug
- long to compile

I nevertheless think that

- std containers are pretty cool
- templating can give and edge on some cases
- sometimes OOP may be a better approach to the Unix rule of representation (sometimes)

However, the pros of C++ do not outweigh its cons for a cloud-first finite-elementish tool. Also, the name C++ is lame.

E.9.2. Fortran?

Because I am not insane (yet). I do not know any sane person that would start writing a piece of software from scratch using Fortran in the 21st century AD.

E.9.3. Python or R?

Python was not designed to perform actual computations but to add another layer so as to ease some (and only some) tasks. The actual computations are written in low-level languages, not in Python (nor Octave, R, etc.) And even if it was, I would not choose a language where scope depends on the indentation.

E.9.4. Go, Rust or Julia?

I don't know them in detail so I cannot tell if any of these languages would be a good fit for FeenoX. Keep in mind that it took me a while to know why not Fortran nor C++ even though there are people that would choose them over C. Maybe something of the sort happens with these new ideas (or not, I don't know).