FeenoX Software Design Specification

Jan/18/2022

Abstract

This Software Design Specifications (SDS) document applies to an imaginary Software Requirement Specifications (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.

Contents

1 Introduction **3**
- 1.1 Objective . . . 4
- 1.2 Scope . . . 5

2 Architecture **12**
- 2.1 Deployment . . . 19
- 2.2 Execution . . . 20
  - 2.2.1 Direct execution . . . 21
- 2.3 Parametric . . . 23
  - 2.3.1 Optimization loops . . . 25
- 2.4 Efficiency . . . 27
- 2.5 Scalability . . . 30
- 2.6 Flexibility . . . 31
- 2.7 Extensibility . . . 36
- 2.8 Interoperability . . . 37

3 Interfaces **43**
- 3.1 Problem input . . . 43
- 3.2 Results output . . . 44

4 Quality assurance **45**
- 4.1 Reproducibility and traceability . . . 45
- 4.2 Automated testing . . . 46
- 4.3 Bug reporting and tracking . . . 46
- 4.4 Verification . . . 47
- 4.5 Validation . . . 48
- 4.6 Documentation . . . 49

A Appendix: Downloading and compiling FeenoX **50**
- A.1 Binary executables . . . 50
- A.2 Source tarballs . . . 51
- A.3 Git repository . . . 52

B Appendix: Rules of UNIX philosophy **54**
B.1 Rule of Modularity . . . 54
B.2 Rule of Clarity . . . 54
B.3 Rule of Composition . . . 54
B.4 Rule of Separation . . . 55
B.5 Rule of Simplicity . . . 55
B.6 Rule of Parsimony . . . 55
B.7 Rule of Transparency . . . 55
B.8 Rule of Robustness . . . 56
B.9 Rule of Representation . . . 56
B.10 Rule of Least Surprise . . . 56
B.11 Rule of Silence . . . 56
B.12 Rule of Repair . . . 56
B.13 Rule of Economy . . . 57
B.14 Rule of Generation . . . 57
B.15 Rule of Optimization . . . 57
B.16 Rule of Diversity . . . 57
B.17 Rule of Extensibility . . . 58

C Appendix: Downloading & compiling **59**
C.1 Quickstart . . . 60
C.2 Detailed configuration and compilation . . . 61
C.2.1 Mandatory dependencies . . . 61
C.2.2 Optional dependencies . . . 62
C.2.3 FeenoX source code . . . 63
C.2.4 Configuration . . . 64
C.2.5 Source code compilation . . . 65
C.2.6 Test suite . . . 67
C.2.7 Installation . . . 67
C.3 Advanced settings . . . 67
C.3.1 Compiling with debug symbols . . . 67
C.3.2 Using a different compiler . . . 67
C.3.3 Compiling PETSc . . . 69

Chapter 1

Introduction

> A computational tool (herein after referred to as *the tool*) specifically designed to be executed in arbitrarily-scalable remote server (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 2022 (and the years to come) are defined.
>
> On the one hand, the tool should allow to solve industrial problems under stringent efficiency (sec. 2.4) and quality (sec. 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, so 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 (sec. 1.1 and sec. 2.7) 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.

Besides noting that software being *free* (regarding freedom, not price) does not imply the same as being *open source*, the requirement is clear in that the tool has to be both *free* and *open source*, a combination which is usually called FOSS. This condition leaves all of the well-known non-free (i.e. incorrectly-called "commercial") finite-element solvers available in the market (NASTRAN, Abaqus, ANSYS, Midas, etc.) out of the tender.

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.
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 the mechanisms available to get, modify, compile, run and document these modifications. 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. 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.

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 they are clearly marked as different from the original and only 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) and what is an aggregate (in which each part be distributed under different licenses) and about usage over a network and the possibility of using AGPL instead of GPL to further enforce freedom (TL;DR: it does not matter for FeenoX), but which are already taken into account in FeenoX licensing scheme.

It should be noted that not only is FeenoX free and open source, but also all of the libraries it depends (and their dependencies) are. It can also be compiled using free and open source build tool chains running over free and open source operating systems. In addition, the FeenoX documentation is licensed under the terms of the GNU Free Documentation License v1.3 (or any later version).

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
> - frequency studies

[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.

> - electromagnetism
> - chemical diffusion
> - process control dynamics
> - computational fluid dynamics
> - ...
>
> on one or more mainstream cloud servers, i.e. computers with hardware and operating systems (futher discussed in sec. 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 sec. 2.7.

The choice of the initial supported features is based on the types of problem that the FeenoX' 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. It is also a first usable version so scope can be bounded. A subsequent road map and release plans can be designed as requested. FeenoX' 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

FeenoX is designed to be developed and executed under GNU/Linux, which is the architecture of more than 95% of the internet servers which we collectively call "the cloud." It should be noted that GNU/Linux is a POSIX-compliant version of UNIX and that FeenoX follows the rules of Unix philosophy (further explained in sec. B) for the actual computational implementation. Besides POSIX, as explained further below in sec. 2.5, FeenoX also uses MPI which is a well-known industry standard for massive execution of parallel processes, both in multi-core hosts and multi-hosts environments. Finally, if performance and/or scalability are not important issues, FeenoX can be run in a (properly cooled) local PC or laptop.

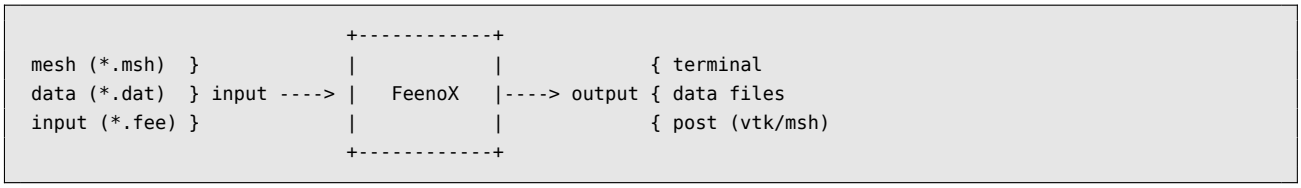
The requirement to run in the cloud and scale up as needed rules out some the open source solver CalculiX. There are some other requirements in the SRS that also leave CalculiX out of the tender.

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 (histori-

> cally called input decks in punched-card days), and/or
> b. written in an high-level interpreted language such as Python or Julia.
>
> 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 (sec. 4.6) the way the input files for a solving particular problem are to be prepared (sec. 3.1) and how the results are to be written (sec. 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.

Indeed, FeenoX is designed to work 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 created 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

i. the quotes above, and
ii. many FEA programs that are still mainstream today

date both from the early 1970s, fifty years later they still

- do not make just only one thing well,
- do complicate old programs by adding new features,
- do not expect the 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 output.

For example, let us consider the famous chaotic Lorenz' 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 ressembles the parameters, inital 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
```

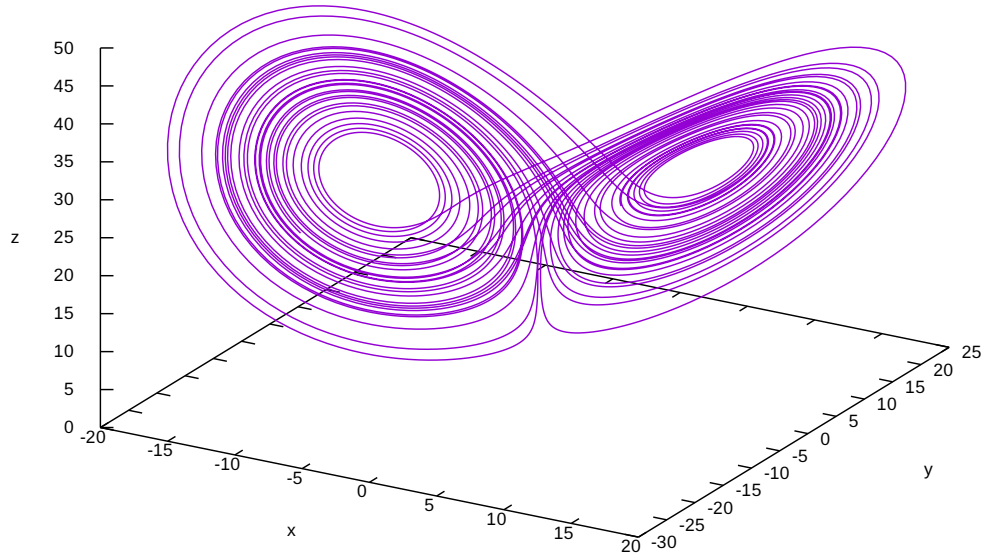


Figure 1.1: The Lorenz attractor solved with FeenoX and drawn with Gnuplot

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' outputs. This is essentially how UNIX filters and pipes work.

As already stated, FeenoX is designed and implemented following the UNIX philosophy in general and Eric Raymond's 17 Unix Rules ([sec:unix]) in particular. One of the main ideas is the *rule of separation* that essentially asks to separate mechanism from policy, that in the computational engineering world translates into separating the frontend from the backend. The usage of FeenoX to compute and of Gnuplot to plot is a clear example of separation. Same idea applies to PDEs, where 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.

From the very beginning, FeenoX is designed as a pure backend which should nevertheless provide appropriate mechanisms for different frontends 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 frontends can radically change (say from a desktop-based point-and-click program to a web-based immersive augmented-reality application) without needing the modify the backend. 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 FOSS third-party libraries such as Metis) for scaling up large problems.

Let us solve the linear elasticity benchmark problem NAFEMS LE10 "Thick plate pressure." Assuming a proper mesh has already been created in Gmsh, note how well the FeenoX input file matches the problem statement:

```
# 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 "sigma_y @ D = " sigmay(2000,0,300) "MPa"
```

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.

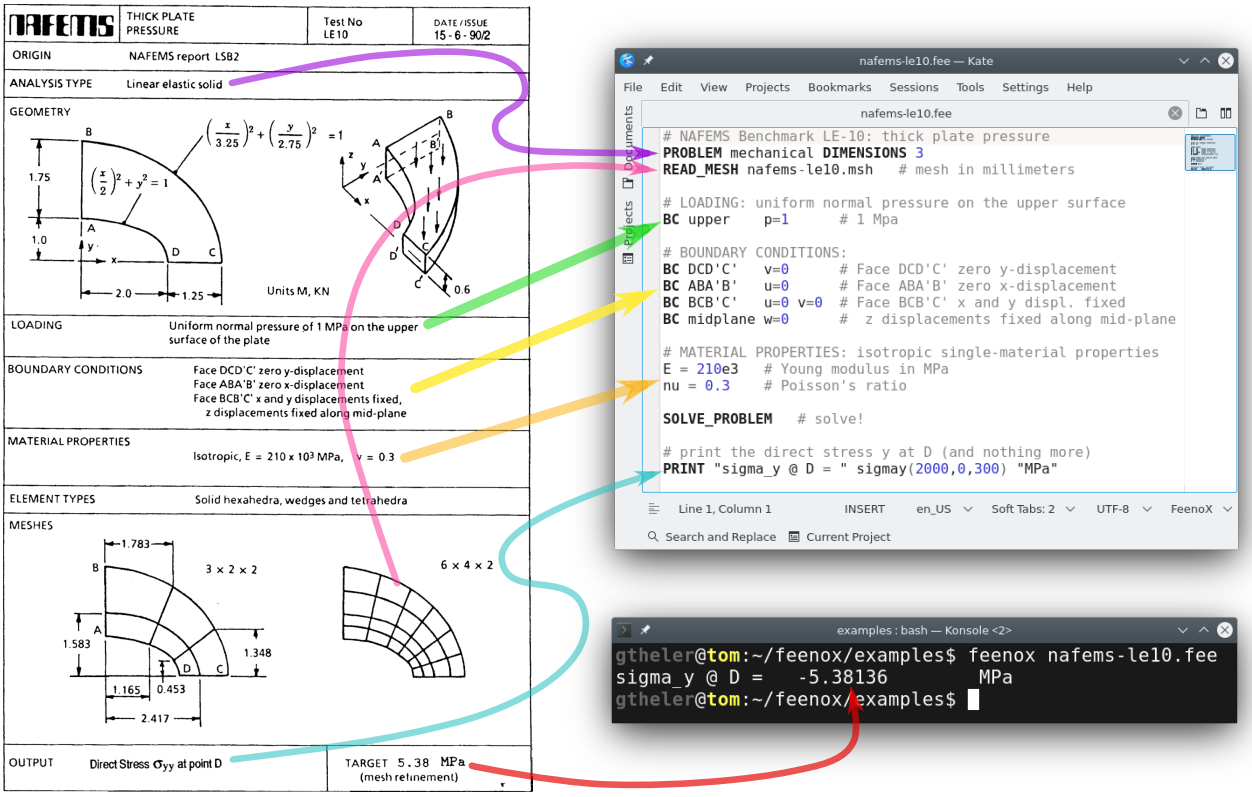


Figure 1.2: The NAFEMS LE10 problem statement and the corresponding FeenoX input

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).

See `https://www.caeplex.com` for a mobile-friendly web-based interface for solving finite elements in the cloud directly from the browser.

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.

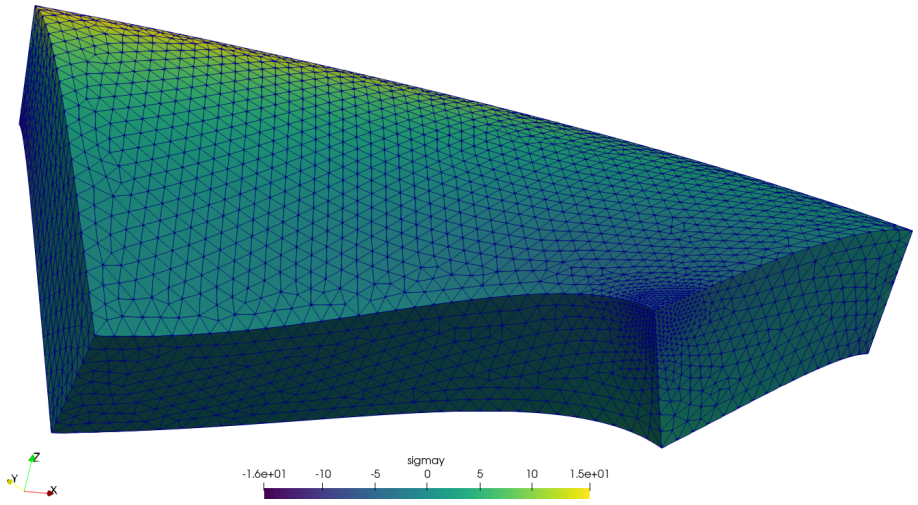


Figure 1.3: Normal stress σ_y refined around point D over 5,000x-warped displacements for LE10 created with Paraview

Chapter 2

Architecture

> The tool must be aimed at being executed unattended on remote servers which are expected to have a mainstream (as of the 202s) architecture regarding operating system (GNU/Linux variants and other UNIX-like OSes) and hardware stack, such as
>
> - a few Intel-compatible CPUs per host
> - a few levels of memory caches
> - a few gigabytes of random-access memory
> - several gigabytes of solid-statee 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.

FeenoX can be seen as a third-system effect, being the third version written from scratch after a first implementation in 2009 and an second one which was far more complex and had far more features circa 2012–2014. The third attempt 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*, 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
- Pyxplot for plotting
- 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 programmed by professional programmers. In particuar, it depends on

- GNU Scientific Library for general mathematics,
- SUNDIALS IDA for ODEs and DAEs,
- PETSc for 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 file and the mesh defining a PDE-casted problem and the mathematical libraries used to solve the discretized equations. This way, FeenoX bounds its scope to do only one thing and to do it well: to build and solve finite-element formulations of thermo-mechanical problems. And it does so on high grounds, both

i. 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, and

ii. 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' main development architecture is Debian GNU/Linux running over 64-bits Intel-compatible processors. All the dependencies are free and/or open source and already available in Debian's official repositories, as explained in sec. 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 plain 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. For its basic mathematical capabilities, FeenoX uses the GNU Scientifc 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 in the industry and academia.

Moreover, PETSc and SLEPc are scalable through the MPI standard (further discussed in sec. 2.5). 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
- Docker/Kubernetes
- AWS/DigitalOcean/Contabo

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 (fig. 2.1).

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 Simpson and you want to solve a maze drawn in a restaurant's placemat, one where both the start and end are known beforehand as show in fig. 2.2. 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.

1. Go to `http://www.mazegenerator.net/`
2. Create a maze
3. Download it in PNG (fig. 2.3a)
4. Perform some conversions
    - PNG → PNM → SVG → DXF → GEO

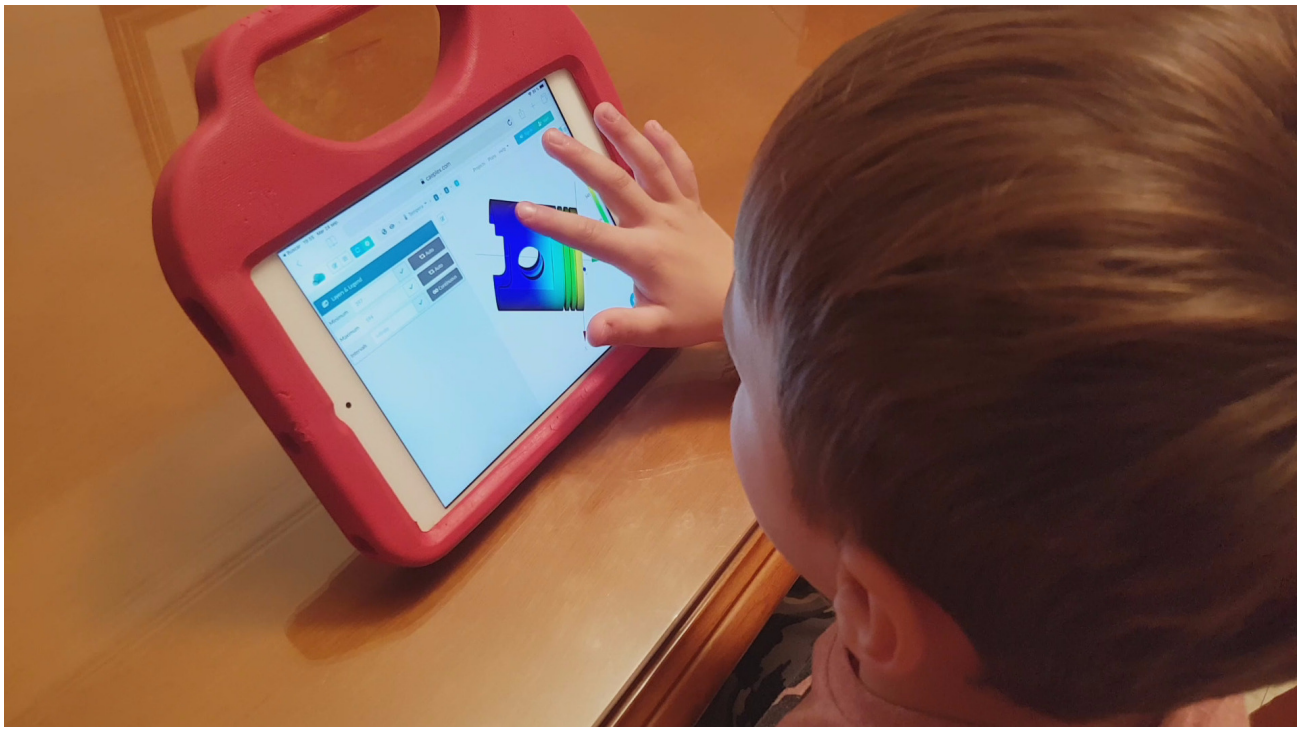


Figure 2.1: The web-based platform CAEplex is mobile-friendly. https://www.youtube.com/watch?v=7KqiMbrSLDc

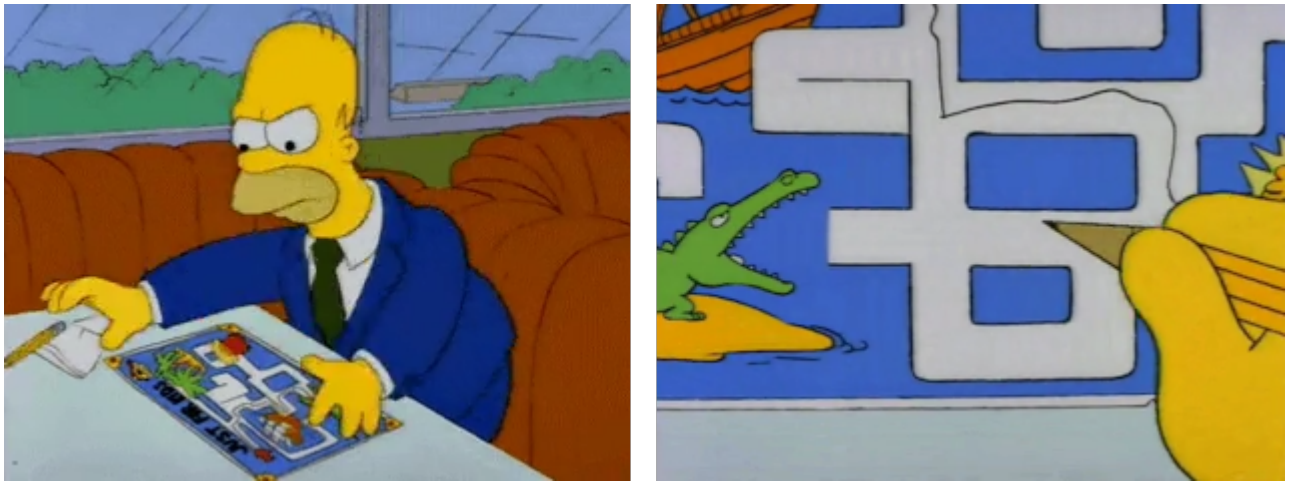
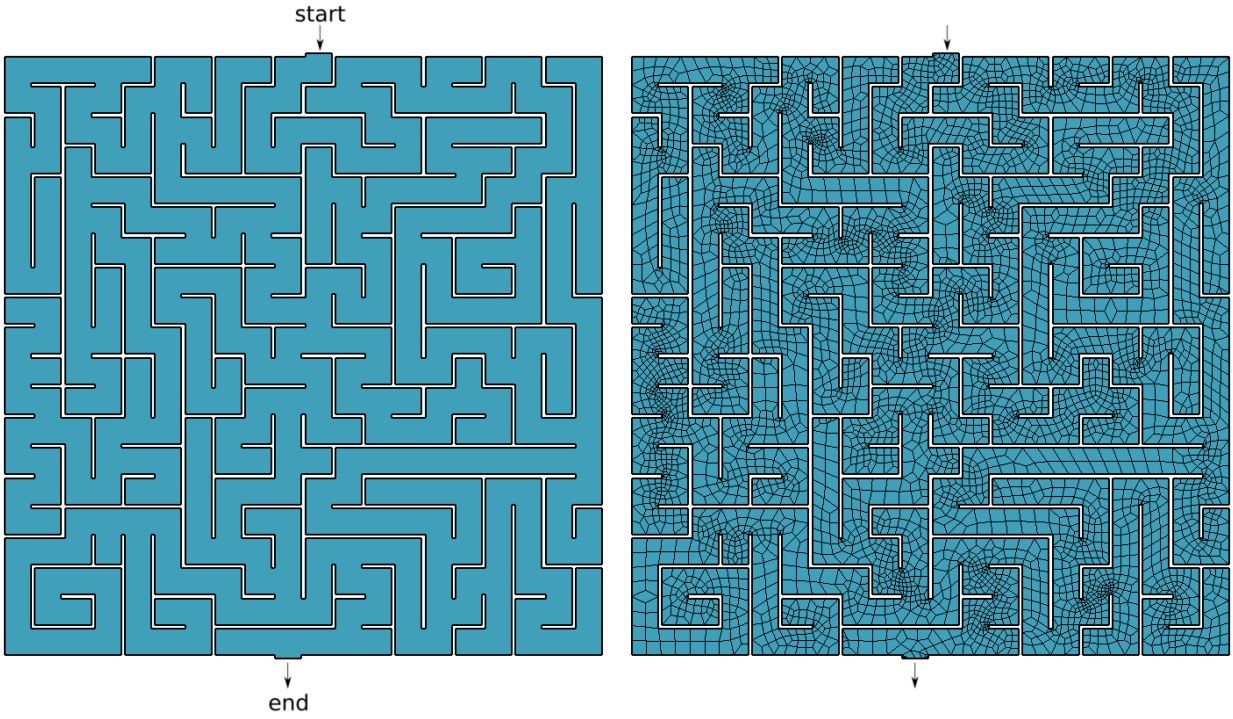
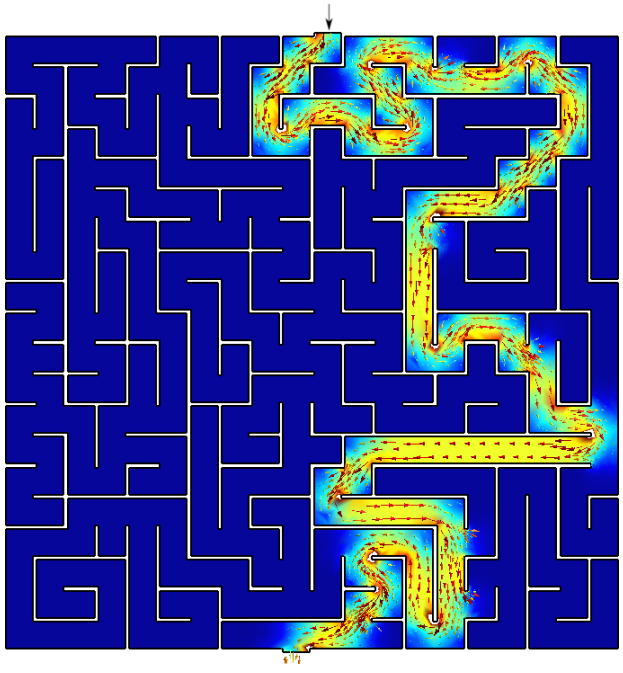


Figure 2.2: Homer trying to solve a maze on a placemat



(a) Bitmapped maze from `https://www.mazegenerator.net` (left) and 2D mesh (right)

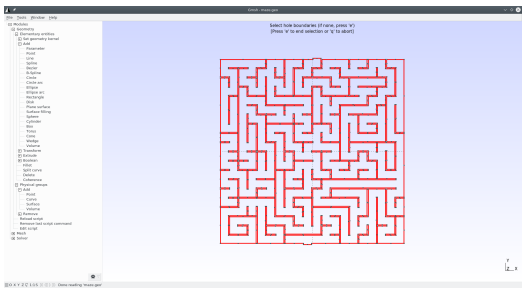


(b) Solution to found by FeenoX (and drawn by Gmsh)

Figure 2.3: Bitmapped, meshed and solved mazes.

```
$ 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 svg -o maze.svg
$ ./svg2dxf maze.svg 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 (fig. 2.3a)

```
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 $\nabla\phi$!

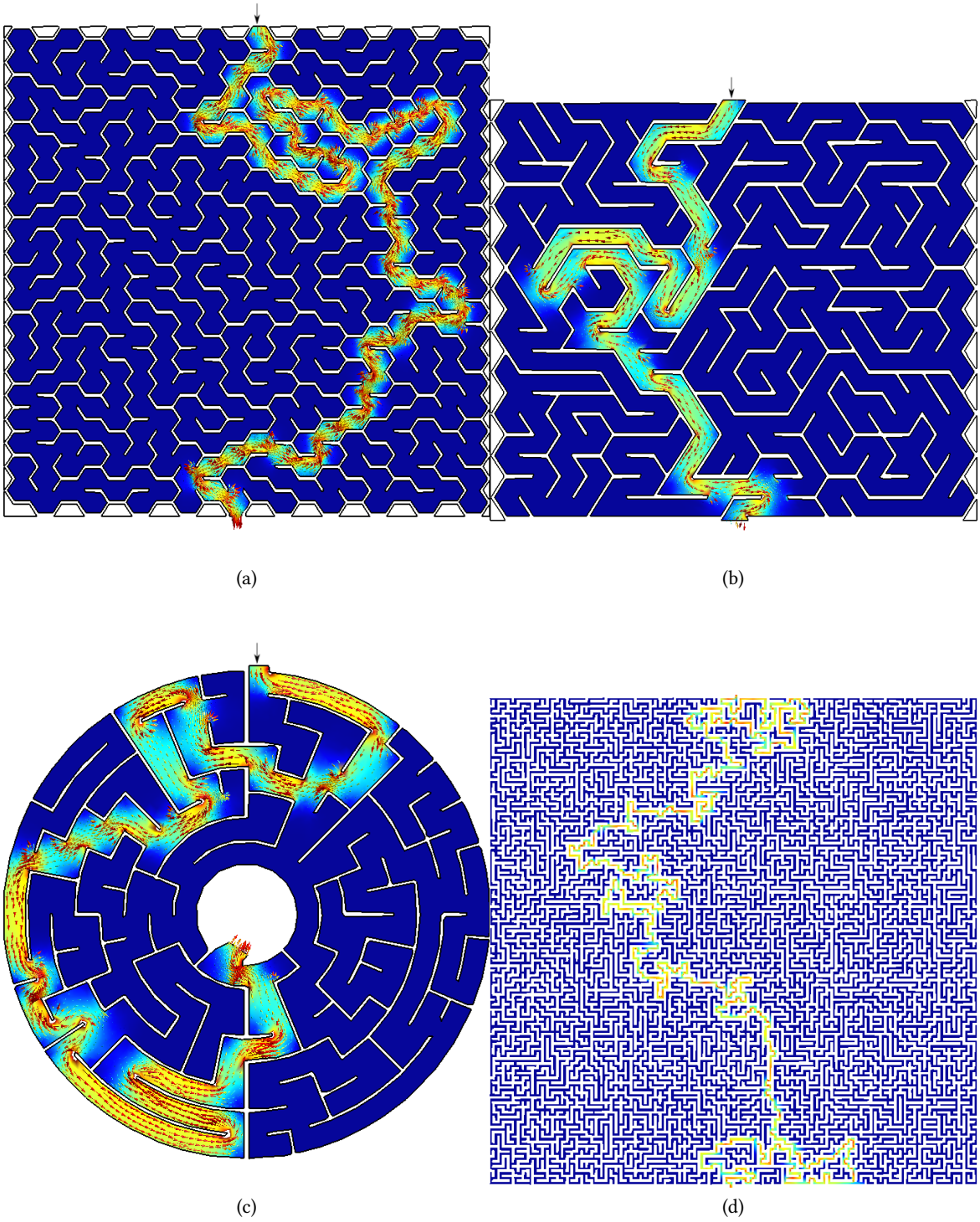


Figure 2.4: Any arbitrary maze (even hand-drawn) can be solved with FeenoX

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 sec. 4.1.

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
- LLVM Clang compiler
- Intel C compiler

FeenoX depends on the four open source libraries stated in sec. 2, with the last three of them being optional. The only mandatory library is the GNU Scientific Library which is part of the 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 with usable binaries for some of the most common architectures—including some non-GNU/Linux variants. These binary distributions contain statically-linked executables that do not need any other shared libraries to be present on the target host, but 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 outweights 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 binaries 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
```

Appendix has sec. C more details about how to download and compile FeenoX. The full documentation contains a compilation guide with further detailed explanations of each of the steps involved. Since all the commands needed to either download a binary executable or to compile from source with customized optimization flags can be automatized, FeenoX can be built into a container such as Docker. This way, deployment and scalability can be customized and tweaked as needed.

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. 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 a mean 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 if 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 FeenoX is designed to run as a file filter (i.e. 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. When invoked without arguments, it prints its version (a through explanation of the versioning scheme is given in sec. 4.1), a one-line description and the usage options:

```
$ feenox
FeenoX v0.1.77-g9325958
a free no-fee no-X uniX-like finite-element(ish) computational engineering tool

usage: feenox [options] inputfile [replacement arguments]

  -h, --help         display usage and commmand-line help and exit
  -v, --version      display brief version information and exit
  -V, --versions     display detailed version information
  -s, --sumarize     list all symbols in the input file and exit

Instructions will be read from standard input if “”- is passed as
inputfile, i.e.

    $ echo "PRINT 2+2" | feenox -
    4

Report bugs at https://github.com/seamplex/feenox or to jeremy@seamplex.com
Feenox home page: https://www.seamplex.com/feenox/
```

The program can also be executed remotely

1. on a server through a SSH session
2. in a container as part of a provisioning script

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`) to more complex mechanisms like writing the status in a shared memory segment.

Regarding its execution, there are three ways of solving problems: direct execution, parametric runs and optimization loops.

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.

2.3 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

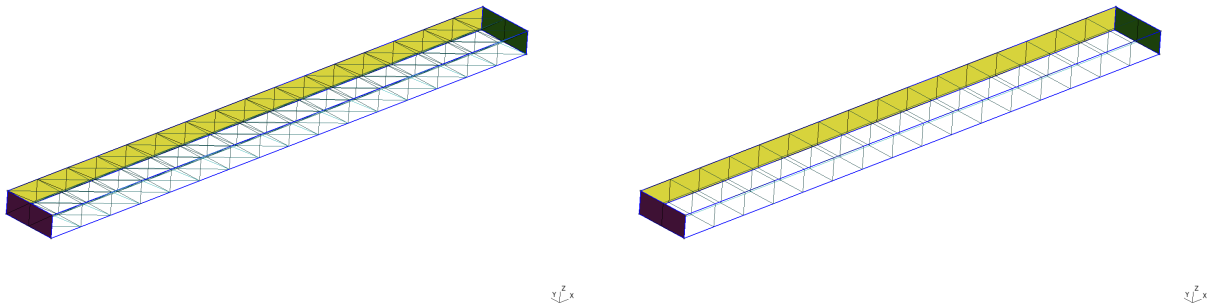
```
$ feenox hello.fee World
Hello World!
$ feenox hello.fee Universe
Hello Universe!
$
```

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"
```

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



(a) Tetrahedra (b) Hexahedra

Figure 2.5: Cantilevered beam meshed with structured tetrahedra and hexahedra

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 Bash script, thanks to the design decision 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 fig. 2.6. 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.

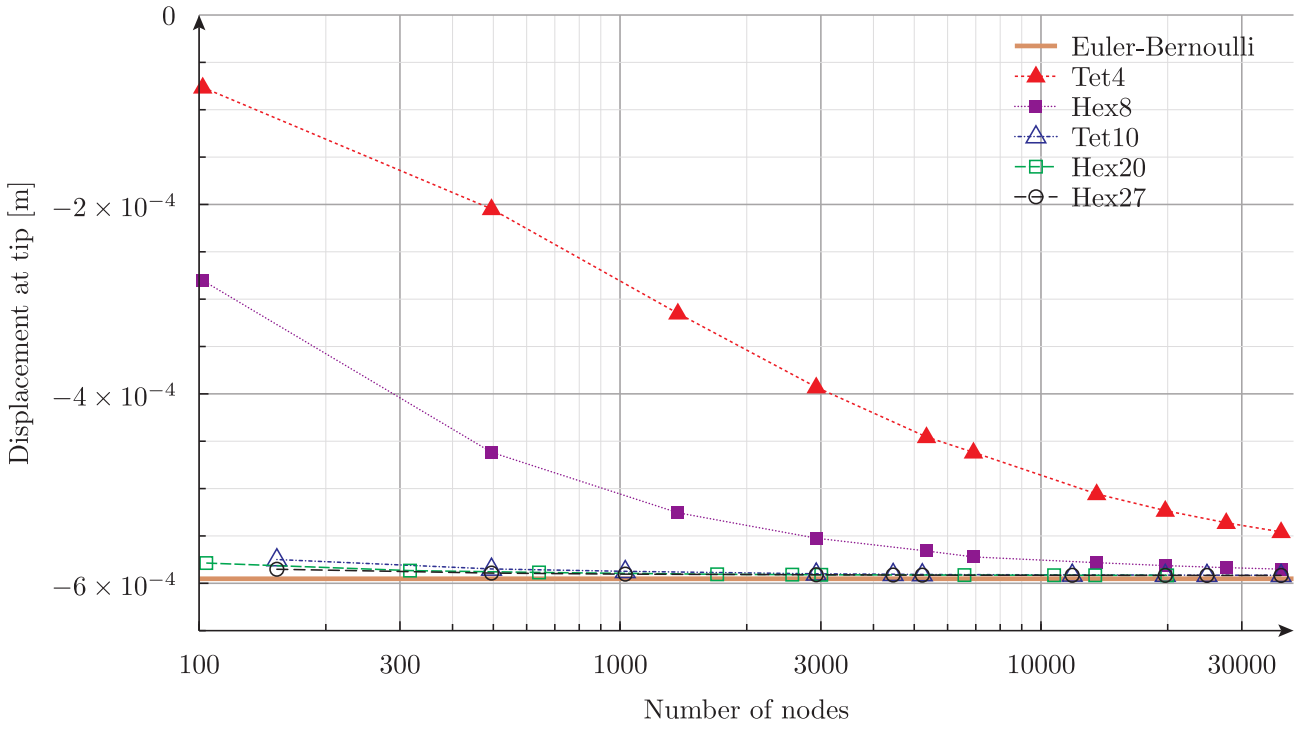


Figure 2.6: Displacement at the free tip of a cantilevered beam vs. number of nodes for different element types

2.3.1 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' 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 (fig. 2.7) 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*) 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]
nu = 0.33
rho = 7829          # in [kg/m^2]

# no BCs! It is a free-free vibration problem
```

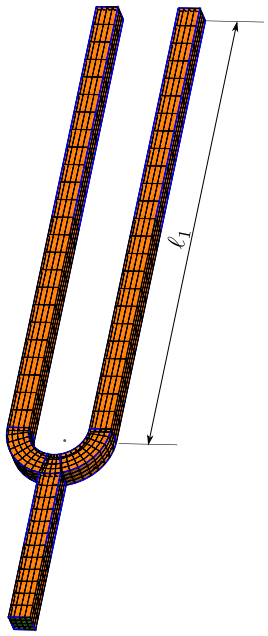


Figure 2.7: What length ℓ_1 is needed so the fork vibrates at 440 Hz?

```
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
$
```

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`.

Fig. 2.8 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.

2.4 Efficiency

> It is mandatory to be able to execute the tool automatically in a remote server. 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 hist The Art

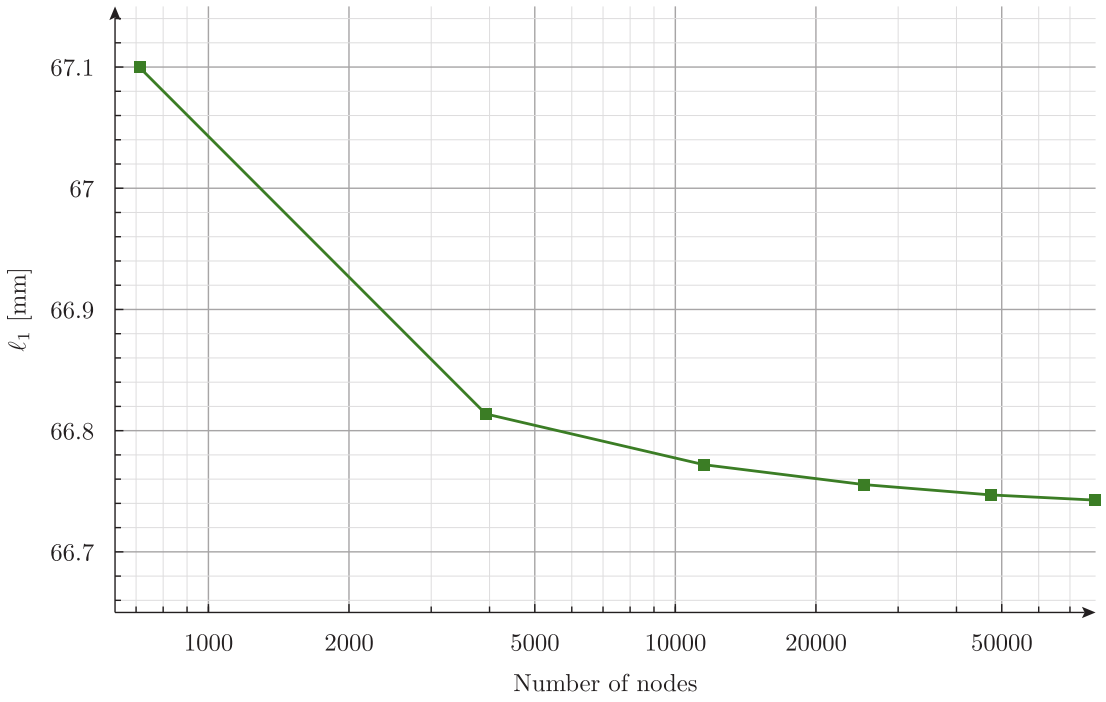
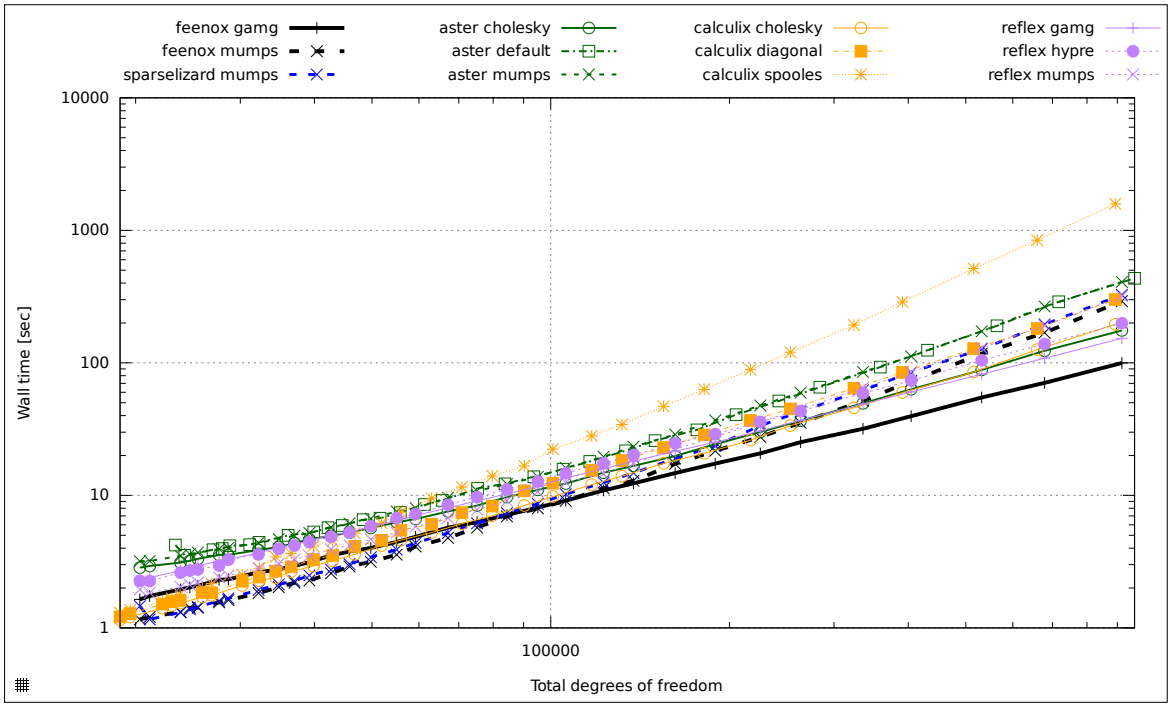


Figure 2.8: Estimated length ℓ_1 needed to get 440 Hz for different mesh refinement levels n

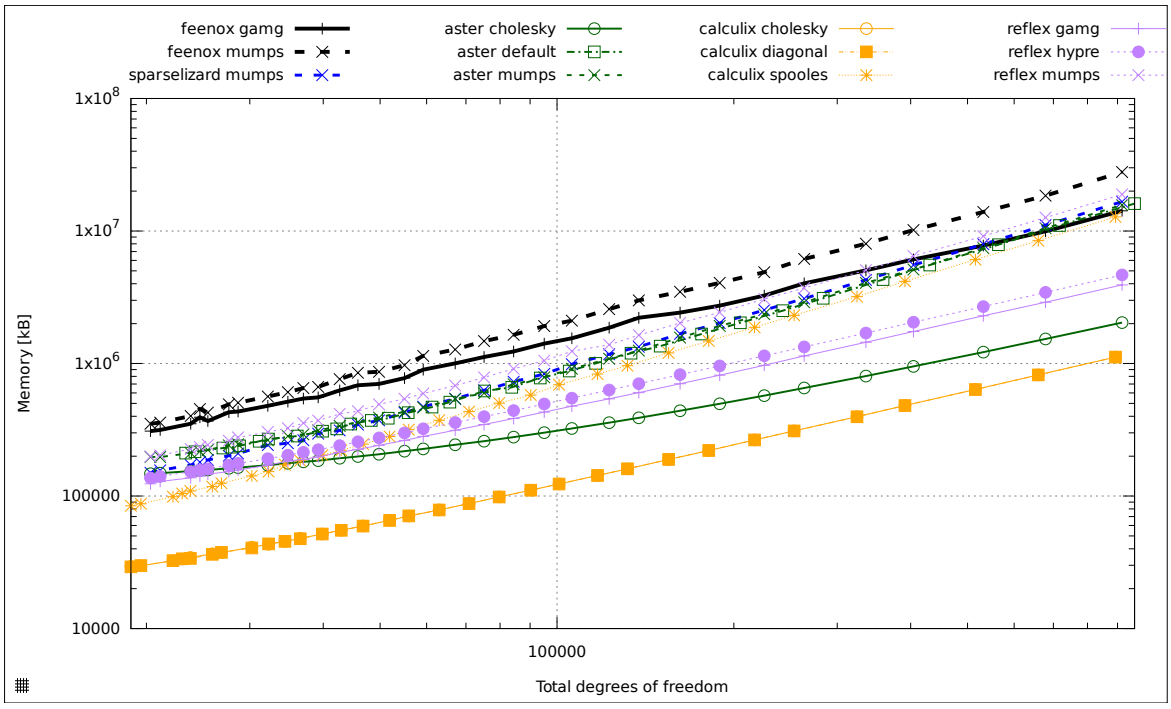
of Computer Programming. Bottom line is that the programmer should not not spend too much time trying to optimize code based on hunches but based on profiling measurements. Yet a disciplined programmer can tell when an algorithm will be way too inefficient (say something that scales up like $O(n^2)$) and how small changes can improve performance (say by understanding how caching levels work). It is also true that usually an improvement in one aspect leads to a deterioration in another one (e.g. decrease in CPU time by caching intermediate results increasing 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. Fig. 2.9 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.

The large memory consumption shown by FeenoX is due to a high level of caching intermediate results. For instance, all the shape functions evaluated at the integration points are computed once when building the stiffness matrix, stored in RAM and then re-used when recovering the gradients of the displacements needed to compute the stresses. There are also a number of non-premature optimization tasks that can improve both the CPU and memory usage that ought to be performed at later stages of the project.



(a) Wall time vs. number of degrees of freedom



(b) Memory vs. number of degrees of freedom

Figure 2.9: 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' script-friendliness using proper calls to `gzip` before and/or after calling the `feenox` binary.

2.5 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 sec. 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).

The time needed to solve a relatively large problem can be reduced by exploiting the fact that most cloud servers (and even laptop computers) have more than one CPU available. There are some tasks that can be split into several processors sharing a common memory address space that will scale up perfectly, such as building the elemental matrices and assembling the global stiffness matrix. There are some other tasks that might not scale perfectly but that nevertheless might (or might not) reduce the overall wall time if split among processors using a common memory space, such as solving the linear system $K \cdot \mathbf{u} = \mathbf{b}$. The usual scheme to parallelize a problem under these conditions is to use the OpenMP framework.

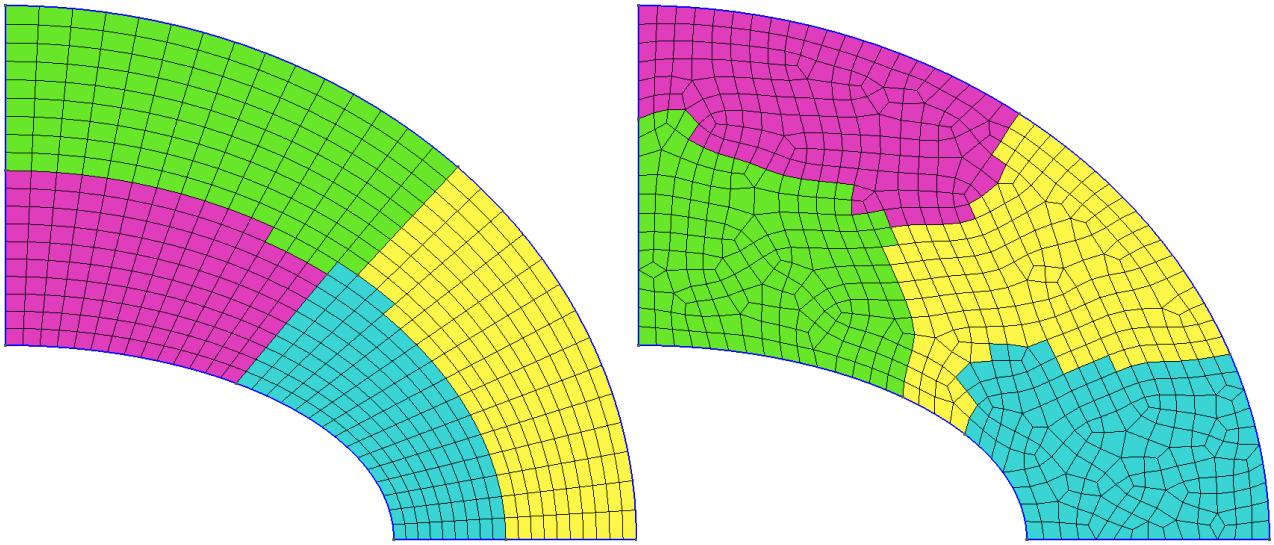
Yet, if the problem is large enough, a server might not have enough physical random-access memory to be able to handle the whole problem. The problem now has to be split among different servers which, in turn, might have several processors each. Some of the processors share the same address space but most of them will only have access to a fraction of the whole global problem data. In these cases, there are no tasks that can scale up perfectly since even when building and assembling the matrices, a processor needs some piece of data which is handled by another processor with a different address space and that has to be conveyed specifically from one process to another one. The usual scheme to parallelize a problem under these conditions is to use the MPI standard and one of its two most well-known implementations, either Open MPI or MPICH.

It might seem that the most effective approach to solve a large problem is to use OpenMP 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 OpenMPI+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.

Most of the overhead of parallelized tasks come from the fact that processes need data stored in other processes that use another memory address space. Therefore, the discretized domain has to be split among processes in such a way as to minimize the number of inter-process communication. This problem, called domain decomposition, can be handled either by the mesher or by the solver itself, usually using a third-part library such as



(a) Structured grid (b) Unstructured grid

Figure 2.10: Partition of the 2D NAFEMS LE1 domain into four different sub-domains computed in Gmsh using Metis.

Metis. It should be noted that the domain decomposition problem does not have a unique solution. On the one hand, it depends on the actual mesh being distributed over parallel processes as illustrated in fig. 2.10. On the other hand, the optimal solution might depend on the kind of topology boundaries to minimize (shared nodes, shared faces) and other subtle options that partitioning libraries allow.

FeenoX relies on Gmsh to perform the domain decomposition (using Metis) and to provide the partitioning information in the mesh file read by the `READ_MESH` keyword.

2.6 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 sec. 2 involves almost ten years of experience in the nuclear industry,[1] 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**. 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

[1] This experience also shaped many of the features that FeenoX has and most of the features is does deliberately not have.

$$\int_0^1 x^{-x}\,dx = \sum_{n=1}^{\infty} n^{-n}$$

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)
```

```
$ feenox sophomore.fee
1.2912861
1.2912860
$
```

Of course most engineering problems will not need explicit integrals (a few of them do, though) but some of them 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."

Flexibility in defining non-trivial material properties is illustrated with the following example, where two non-squares made of different dimensional materials are juxtaposed in thermal contact and subject to different boundary conditions at each of the fours sides (fig. 2.11).

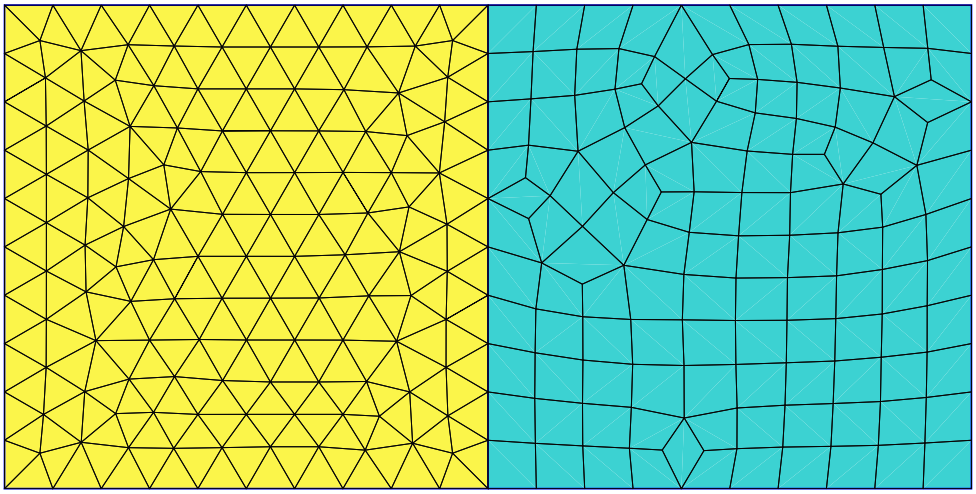


Figure 2.11: Two non-dimensional 1×1 squares each in thermal contact made of different materials.

The yellow square is made of a certain material with a conductivity that depends algebraically on 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

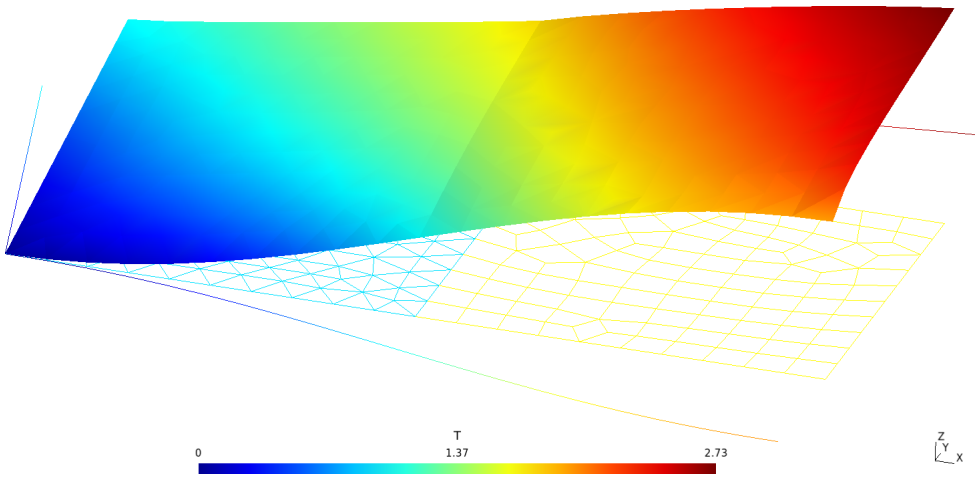
$$\begin{cases} T(x,y) = y & \text{at the left edge } y = 0 \\ T(x,y) = 1 - \cos\left(\frac{1}{2}\pi \cdot x\right) & \text{at the bottom edge } x = 0 \\ q'(x,y) = 2 - y & \text{at the right edge } x = 2 \\ q'(x,y) = 1 & \text{at the top edge } y = 1 \end{cases}$$

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:

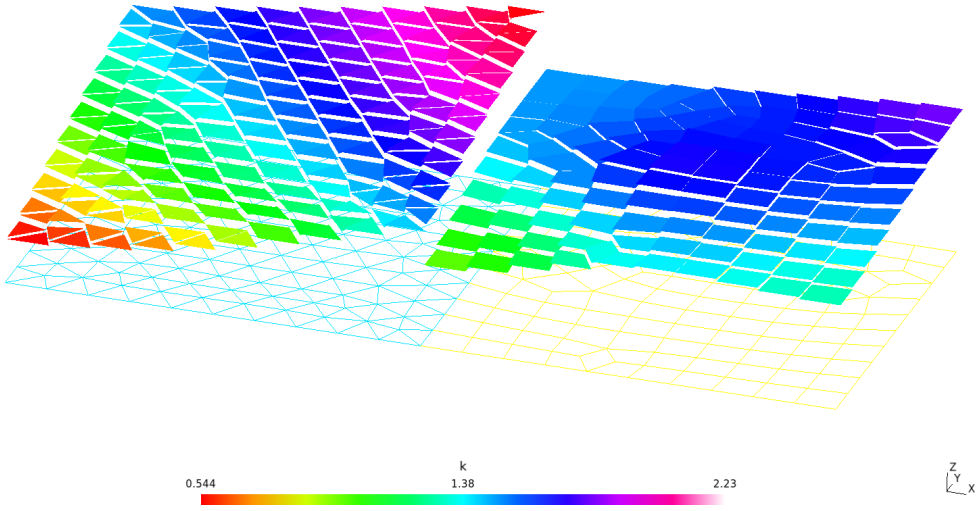
```
PROBLEM thermal 2d            # heat conduction in two dimensions
READ_MESH two-squares.msh

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).



(a) Temperature defined at nodes



(b) Conductivity defined at cells

Figure 2.12: Temperature (main result) and conductivity for the two-squares thermal problem.

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 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.

$$T(x=0) = 0 \text{ ℃}$$

and to a transient temperature

$$T(x = 0.1 \text{ m}, t) = 100 \text{ ℃} \cdot \sin\left(\frac{\pi \cdot t}{40 \text{ s}}\right)$$

at the other side. There is zero internal heat generation, at $t = 0$ all temperature is equal to 0℃ (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 $T(0.08 \text{ m}, 32 \text{ s}) = 36.60$ ℃.

```
PROBLEM heat 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
$
```

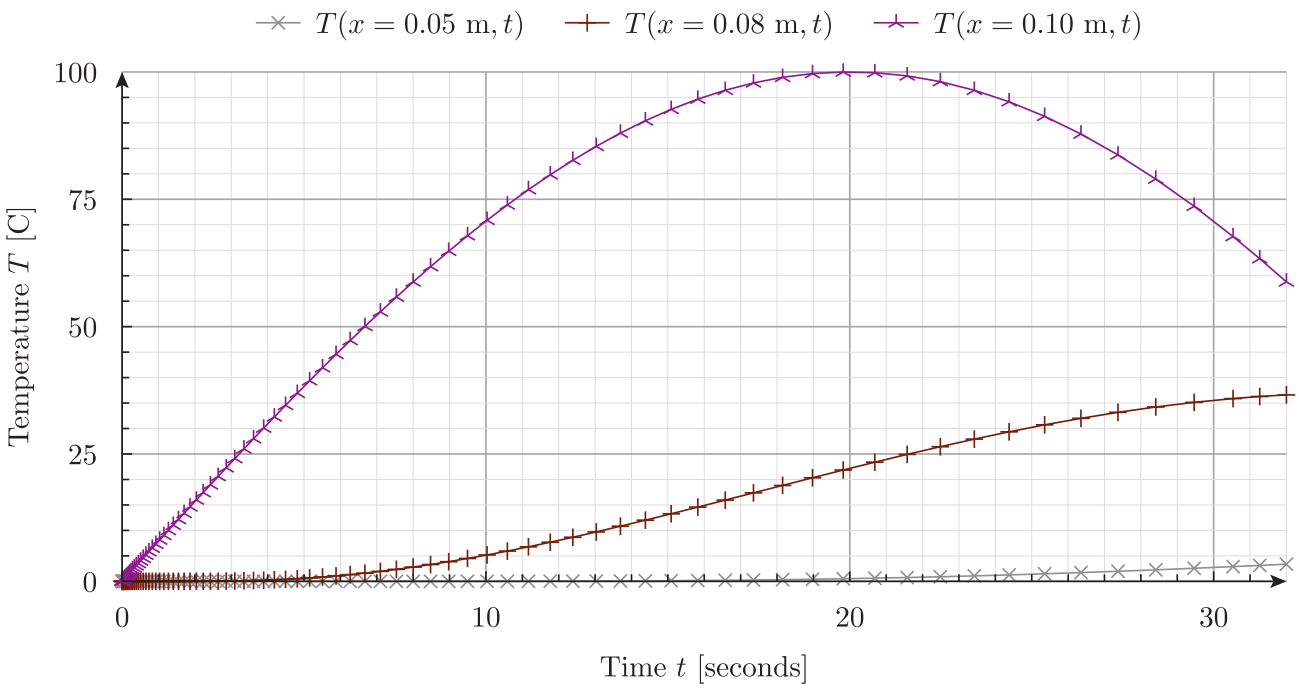


Figure 2.13: Temperature vs. time at three axial locations for the NAFEMS T3 benchmark

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 sec. 3.1.

2.7 Extensibility

> It should be possible to add other PDE-casted problem types (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.

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 routines are called to

1. initialize and set up solver options (steady-state/transient, linear/non-linear, regular/eigenproblem, 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`

is a separate directory under `src/pdes` that implements these "virtual" methods that are resolved at runtime when parsing the main input file. 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`).

As already discussed in sec. 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 additional problem types, 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 using one of the existing ones as a template or by creating a new directory from scratch, without asking anybody of 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. This is 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. The authors own the copyright of the additional code. Yet, if they want to distribute the modified version they have to do it under 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.[2] That is to say, free software ought to remain free.

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, etc.) that are resolved at run time. Following the same principle, new models might be added by adding new routines and resolving them depending on the user's input.

2.8 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.

Sec. 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:

[2] Even better, these authors should ask to merge their contributions into FeenoX' main code base.

- Kate (with syntax highlighting): fig. 1.2
- Gmsh (both as a mesher and a post-processor): figs. 2.3, 2.4, 2.5, 2.7, 2.10, 2.11, 2.12
- Paraview: fig. 1.3
- Gnuplot: figs. 1.1, 2.9
- Pyxplot: figs. 2.6, 2.8, 2.13

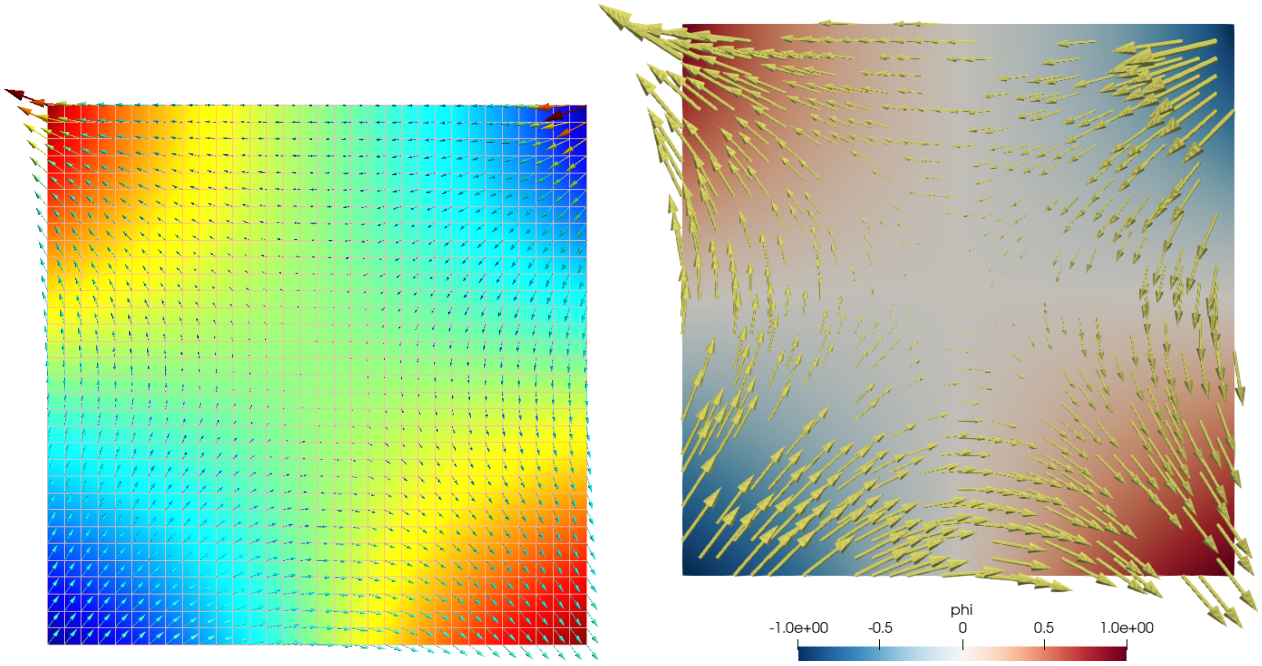
To illustrate both the filter approach, consider the following input file that solves Laplace's equation $\nabla^2\phi = 0$ on a square with some space-dependent boundary conditions. Either Gmsh or Paraview can be used to post-process the results:

$$\begin{cases} \phi(x,y) = +y & \text{for } x = -1 \text{ (left)} \\ \phi(x,y) = -y & \text{for } x = +1 \text{ (right)} \\ \nabla\phi \cdot \hat{\mathbf{n}} = \sin\left(\frac{\pi}{2} \cdot x\right) & \text{for } y = -1 \text{ (bottom)} \\ \nabla\phi \cdot \hat{\mathbf{n}} = 0 & \text{for } y = +1 \text{ (top)} \end{cases}$$

```
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 sec. 3.2), a.k.a. "no PRINT, no OUTPUT" whose corollary is the UNIX *rule of silence*. 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 GFM 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/fino/doc/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 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
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)
```



(a) Post-processed with Gmsh

(b) Post-processed with Paraview

Figure 2.14: Laplace's equation solved with FeenoX

```
# 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 tbl. 2.2.

Table 2.2: Comparison of analytical and numerical frequencies, in Hz

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
4	1573	1576	0.24
5	2596	2606	0.38

- UNIX
- POSIX
- shmem
- mpi
- moustache

It should be noted that all of the programs and tools mentioned to be interoperable with FeenoX are free and open source software.

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 paste 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 2.15: Results of the same fatigue problem solved using two different philosophies.

Chapter 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 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.

3.1 Problem input

> The problem should be completely defined by one or more input files. These input files might be
>
> - particularly formatted files to be read by the tool in an *ad-hoc* way, and/or
> - source files for interpreted languages which can call the tool through and API or equivalent method, and/or
> - any other method that can fulfill the requirements described so far.
>
> Preferably, these input files should be plain ASCII file in order to be tracked by distributed control version systems such as Git. If the tool provides an API for an interpreted language such as Python, 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 acceptable if only basic usage can be achieved through the usage of graphical interfaces to ease basic usage at least. Complex problems involving non-trivial material properties and boundary conditions might Notwithstanding the suggestion above, it is expected that

dar ejemplos comparar con https://cofea.readthedocs.io/en/latest/benchmarks/004-eliptic-membrane/tested-codes.html

macro-friendly inputs, rule of generation

Simple problems should need simple inputs.

English-like input. Nouns are definitions, verbs are instructions.

Similar problems should need similar inputs.

thermal slab steady state and transient

1d neutron

VCS tracking, example with hello world.

API in C?

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.

JSON/YAML, state of the art open post-processing formats. Mobile & web-friendly.

Common and preferably open-source formats.

100% user-defined output with PRINT, rule of silence rule of economy, i.e. no RELAP yaml/json friendly outputs vtk (vtu), gmsh, frd?

90% is serial (vtk), no need to complicate due to 10%

Chapter 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

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 sec. 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 sec. 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.

simple <-> simple

similar <-> Similar

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.

make check

regressions, example of the change of a sign

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 (unxepected run-time errors, segmentation faults, etc.)
> - return a correct result

github

mailing listings

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 analysed 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 [@sandia-mms] 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 sec. 1 and well documented as required in sec. 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 sec. 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.

open source (really, not like CCX -> mostrar ejemplo) GPLv3+ free Git + gitlab, github, bitbucket

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.

already done for Fino

hip implant, 120+ pages, ASME, cases of increasing complexity

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 (sec. 2.7), 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 (sec. 4.4) 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 convections

it's not compact, but almost! 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. unix man page markdown + pandoc = html, pdf, texinfo

Appendix A

Appendix: Downloading and compiling FeenoX

A.1 Binary executables

Browse to https://www.seamplex.com/feenox/dist/ and check what the latest version for your architecture is. Then do

```
wget https://www.seamplex.com/feenox/dist/linux/feenox-v0.1.59-gbf85679-linux-amd64.tar.gz
tar xvzf feenox-v0.1.59-gbf85679-linux-amd64.tar.gz
```

You'll have the binary under `bin` and examples, documentation, manpage, etc under `share`. Copy `bin/feenox` into somewhere in the `PATH` and that will be it. If you are root, do

```
sudo cp feenox-v0.1.59-gbf85679-linux-amd64/bin/feenox /usr/local/bin
```

If you are not root, the usual way is to create a directory `$HOME/bin` and add it to your local path. If you have not done it already, do

```
mkdir -p $HOME/bin
echo 'expot PATH=$PATH:$HOME/bin' >> .bashrc
```

Then finally copy `bin/feenox` to `$HOME/bin`

```
cp feenox-v0.1.59-gbf85679-linux-amd64/bin/feenox $HOME/bin
```

Check if it works by calling `feenox` from any directory (you might need to open a new terminal so `.bashrc` is re-read):

```
$ feenox
FeenoX v0.1.67-g8899dfd-dirty
a free no-fee no-X uniX-like finite-element(ish) computational engineering tool
```

```
usage: feenox [options] inputfile [replacement arguments]

  -h, --help         display usage and commmand-line help and exit
  -v, --version      display brief version information and exit
  -V, --versions     display detailed version information
  -s, --sumarize     list all symbols in the input file and exit

Instructions will be read from standard input if “”- is passed as
inputfile, i.e.

    $ echo "PRINT 2+2" | feenox -
    4

Report bugs at https://github.com/seamplex/feenox or to jeremy@seamplex.com
Feenox home page: https://www.seamplex.com/feenox/
$
```

A.2 Source tarballs

To compile the source tarball, proceed as follows. This procedure does not need `git` nor `autoconf` but a new tarball has to be downloaded each time there is a new FeenoX version.

1. Install mandatory dependencies

   ```
   sudo apt-get install gcc make libgsl-dev
   ```

   If you cannot install `libgsl-dev`, you can have the `configure` script to download and compile it for you. See point 4 below.

2. Install optional dependencies (of course these are *optional* but recommended)

   ```
   sudo apt-get install libsundials-dev petsc-dev slepc-dev
   ```

3. Download and uncompress FeenoX source tarball. Browse to https://www.seamplex.com/feenox/dist/src/ and pick the latest version:

   ```
   wget https://www.seamplex.com/feenox/dist/src/feenox-v0.1.66-g1c4b17b.tar.gz
   tar xvzf feenox-v0.1.66-g1c4b17b.tar.gz
   ```

4. Configure, compile & make

   ```
   cd feenox-v0.1.66-g1c4b17b
   ./configure
   make -j4
   ```

   If you cannot (or do not want) to use `libgsl-dev` from a package repository, call `configure` with `--enable-download-gsl`:

```
./configure --enable-download-gsl
```

If you do not have Internet access, get the tarball manually, copy it to the same directory as `configure` and run again.

5. Run test suite (optional)

```
make check
```

6. Install the binary system wide (optional)

```
sudo make install
```

A.3 Git repository

To compile the Git repository, proceed as follows. This procedure does need `git` and `autoconf` but new versions can be pulled and recompiled easily.

1. Install mandatory dependencies

```
sudo apt-get install gcc make git automake autoconf libgsl-dev
```

If you cannot install `libgsl-dev` but still have `git` and the build toolchain, you can have the `configure` script to download and compile it for you. See point 4 below.

2. Install optional dependencies (of course these are *optional* but recommended)

```
sudo apt-get install libsundials-dev petsc-dev slepc-dev
```

3. Clone Github repository

```
git clone https://github.com/seamplex/feenox
```

4. Boostrap, configure, compile & make

```
cd feenox
./autogen.sh
./configure
make -j4
```

If you cannot (or do not want) to use `libgsl-dev` from a package repository, call `configure` with `--enable-download-gsl`:

```
./configure --enable-download-gsl
```

If you do not have Internet access, get the tarball manually, copy it to the same directory as `configure` and run again.

5. Run test suite (optional)

```
make check
```

6. Install the binary system wide (optional)

```
sudo make install
```

To stay up to date, pull and then autogen, configure and make (and optionally install):

```
git pull
./autogen.sh; ./configure; make -j4
sudo make install
```

Appendix B

Appendix: Rules of UNIX philosophy

B.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 uses third-party high-quality libraries
  - GNU Scientific Library
  - SUNDIALS
  - PETSc
  - SLEPc

B.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.

- Example two squares in thermal contact.
- LE10 & LE11: a one-to-one correspondence between the problem text and the FeenoX input.

B.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.

- FeenoX uses meshes created by a separate mesher (i.e. Gmsh).
- FeenoX writes data that has to be plotted or post-processed by other tools (Gnuplot, Gmsh, Paraview, etc.).

- ASCII output is 100% controlled by the user so it can be tailored to suit any other programs' input needs such as AWK filters to create LaTeX tables.

B.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 does not include a GUI, but it is GUI-friendly.

B.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.

- Simple problems need simple input.
- Similar problems need similar inputs.
- English-like self-evident input files matching as close as possible the problem text.
- If there is a single material there is no need to link volumes to properties.

B.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.

- Parametric and/or optimization runs have to be driven from an outer script (Bash, Python, etc.)

B.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.

- Written in C99
- 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.

B.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.

B.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.

B.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.

- 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
```

B.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.

- No `PRINT` (or `WRITE_MESH`), no output.

B.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 can be user controled, they can be fatal or ignored.

B.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.

- Output is 100% user-defined so the desired results is 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.
- Example: LE10 & LE11.

B.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.

- Inputs are M4-like-macro friendly.
- Parametric runs can be done from scripts through command line arguments expansion.
- Documentation is created out of simple Markdown sources and assembled as needed.

B.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.

- Premature optimization is the root of all evil
- We are still building. We will optimize later.
  - Code optimization: TODO
  - Parallelization: TODO
  - Comparison with other tools: TODO

B.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.

- Either Gmsh or Paraview can be used to post-process results.
- Other formats can be added.

B.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.

- FeenoX is GPLv3+. The '+' is for the future.
- Each PDE has a separate source directory. Any of them can be used as a template for new PDEs, especially `laplace` for elliptic operators.

Appendix C

Appendix: Downloading & compiling

FeenoX is distributed under the terms of the GNU General Public License version 3 or (at your option) any later version. See licensing below for details.

GNU/Linux binaries	`https://www.seamplex.com/feenox/dist/linux`
Windows binaries	`https://www.seamplex.com/feenox/dist/windows`
Source tarballs	`https://www.seamplex.com/feenox/dist/src`
Github repository	`https://github.com/seamplex/feenox/`

- Be aware that FeenoX is a backend. It **does not have a GUI**. Read the documentation, especially the description and the FAQs. Ask for help on the GitHub discussions page.
- Binaries are provided as statically-linked executables for convenience. They do not support MUMPS nor MPI and have only basic optimization flags. Please compile from source for high-end applications. See detailed compilatation instructions.
- 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`

These detailed compilation instructions are aimed at `amd64` Debian-based GNU/Linux distributions. The compilation procedure follows POSIX, so it should work in other operating systems and architectures as well. Distributions not using `apt` for packages (i.e. `yum`) should change the package installation commands (and possibly the package names). The instructions should also work for in MacOS, although the `apt-get` commands should be replaced by `brew` or similar. Same for Windows under Cygwin, the packages should be installed through the Cygwin installer. WSL was not tested, but should work as well.

C.1 Quickstart

Note that the quickest way to get started is to get an already-compiled statically-linked binary executable. Follow these instructions if that option is not suitable for your workflow.

On a GNU/Linux box (preferably Debian-based), follow these quick steps. See next section for detailed explanations.

To compile the Git repository, proceed as follows. This procedure does need `git` and `autoconf` but new versions can be pulled and recompiled easily.

1. Install mandatory dependencies

   ```
   sudo apt-get install gcc make git automake autoconf libgsl-dev
   ```

   If you cannot install `libgsl-dev` but still have `git` and the build toolchain, you can have the `configure` script to download and compile it for you. See point 4 below.

2. Install optional dependencies (of course these are *optional* but recommended)

   ```
   sudo apt-get install libsundials-dev petsc-dev slepc-dev
   ```

3. Clone Github repository

   ```
   git clone https://github.com/seamplex/feenox
   ```

4. Boostrap, configure, compile & make

   ```
   cd feenox
   ./autogen.sh
   ./configure
   make -j4
   ```

   If you cannot (or do not want) to use `libgsl-dev` from a package repository, call `configure` with `--enable-download-gsl`:

   ```
   ./configure --enable-download-gsl
   ```

   If you do not have Internet access, get the tarball manually, copy it to the same directory as `configure` and run again.

5. Run test suite (optional)

   ```
   make check
   ```

6. Install the binary system wide (optional)

   ```
   sudo make install
   ```

To stay up to date, pull and then autogen, configure and make (and optionally install):

```
git pull
./autogen.sh; ./configure; make -j4
sudo make install
```

C.2 Detailed configuration and compilation

The main target and development environment is Debian GNU/Linux, although it should be possible to compile FeenoX in any free GNU/Linux variant (and even the in non-free MacOS and Windows). As per the SRS, all dependencies have to be available on mainstream GNU/Linux distributions. But they can also be compiled from source in case the package repositories are not available or customized compilation flags are needed (i.e. optimization or debugging settings).

All the dependencies are free and open source software. PETSc/SLEPc also depend on other mathematical libraries to perform particular operations such as linear algebra. These extra dependencies can be either free (such as LAPACK) or non-free (such as MKL), but there is always at least one combination of a working setup that involves only free and open source software which is compatible with FeenoX licensing terms (GPLv3+). See the documentation of each package for licensing details.

C.2.1 Mandatory dependencies

FeenoX has one mandatory dependency for run-time execution and the standard build toolchain for compilation. It is written in C99 so only a C compiler is needed, although `make` is also required. Free and open source compilers are favored. The usual C compiler is `gcc` but `clang` can also be used. Nevertheless, the non-free `icc` has also been tested.

Note that there is no need to have a Fortran nor a C++ compiler to build FeenoX. They might be needed to build other dependencies (such as PETSc), but not to compile FeenoX with all the dependencies installed from package repositories. In case the build toolchain is not already installed, do so with

```
sudo apt-get install gcc make
```

If the source is to be fetched from the Git repository then of course `git` is needed but also `autoconf` and `automake` since the `configure` script is not stored in the Git repository but the `autogen.sh` script that bootstraps the tree and creates it. So if instead of compiling a source tarball one wants to clone from GitHub, these packages are also mandatory:

```
sudo apt-get install git automake autoconf
```

Again, chances are that any existing GNU/Linux box has all these tools already installed.

C.2.1.1 The GNU Scientific Library

The only run-time dependency is GNU GSL (not to be confused with Microsoft GSL). It can be installed with

```
sudo apt-get install libgsl-dev
```

In case this package is not available or you do not have enough permissions to install system-wide packages, there are two options.

1. Pass the option `--enable-download-gsl` to the `configure` script below.
2. Manually download, compile and install GNU GSL

If the `configure` script cannot find both the headers and the actual library, it will refuse to proceed. Note that the FeenoX binaries already contain a static version of the GSL so it is not needed to have it installed in order to run the statically-linked binaries.

C.2.2 Optional dependencies

FeenoX has three optional run-time dependencies. It can be compiled without any of these but functionality will be reduced:

- SUNDIALS provides support for solving systems of ordinary differential equations (ODEs) or differential-algebraic equations (DAEs). This dependency is needed when running inputs with the `PHASE_SPACE` keyword.
- PETSc provides support for solving partial differential equations (PDEs). This dependency is needed when running inputs with the `PROBLEM` keyword.
- SLEPc provides support for solving eigen-value problems in partial differential equations (PDEs). This dependency is needed for inputs with `PROBLEM` types with eigen-value formulations such as `modal` and `neutron_transport`.

In absence of all these, FeenoX can still be used to

- solve general mathematical problems such as the ones to compute the Fibonacci sequence,
- operate on functions, either algebraically or point-wise interpolated,
- read, operate over and write meshes,
- etc.

These optional dependencies have to be installed separately. There is no option to have `configure` to download them as with `--enable-download-gsl`.

C.2.2.1 SUNDIALS

SUNDIALS is a SUite of Nonlinear and DIfferential/ALgebraic equation Solvers. It is used by FeenoX to solve dynamical systems casted as DAEs with the keyword `PHASE_SPACE`, like the Lorenz system.

Install either by doing

```
sudo apt-get install libsundials-dev
```

or by following the instructions in the documentation.

C.2.2.2 PETSc

The Portable, Extensible Toolkit for Scientific Computation, pronounced PET-see (/ˈpɛt-siː/), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It is used by FeenoX to solve PDEs with the keyword `PROBLEM`, like thermal conduction on a slab.

Install either by doing

```
sudo apt-get install petsc-dev
```

or by following the instructions in the documentation.

Note that

- Configuring and compiling PETSc from scratch might be difficult the first time. It has a lot of dependencies and options. Read the official documentation for a detailed explanation.
- There is a huge difference in efficiency between using PETSc compiled with debugging symbols and with optimization flags. Make sure to configure `--with-debugging=0` for FeenoX production runs and leave the debugging symbols (which is the default) for development only.
- FeenoX needs PETSc to be configured with real double-precision scalars. It will compile but will complain at run-time when using complex and/or single or quad-precision scalars.
- FeenoX honors the `PETSC_DIR` and `PETSC_ARCH` environment variables when executing `configure`. If these two do not exist or are empty, it will try to use the default system-wide locations (i.e. the `petsc-dev` package).

C.2.2.3 SLEPc

The Scalable Library for Eigenvalue Problem Computations, is a software library for the solution of large scale sparse eigenvalue problems on parallel computers. It is used by FeenoX to solve PDEs with the keyword `PROBLEM` that need eigen-value computations, such as modal analysis of a cantilevered beam.

Install either by doing

```
sudo apt-get install slepc-dev
```

or by following the instructions in the documentation.

Note that

- SLEPc is an extension of PETSc so the latter has to be already installed and configured.
- FeenoX honors the `SLEPC_DIR` environment variable when executing `configure`. If it does not exist or is empty it will try to use the default system-wide locations (i.e. the `slepc-dev` package).
- If PETSc was configured with `--download-slepc` then the `SLEPC_DIR` variable has to be set to the directory inside `PETSC_DIR` where SLEPc was cloned and compiled.

C.2.3 FeenoX source code

There are two ways of getting FeenoX' source code:

1. Cloning the GitHub repository at https://github.com/seamplex/feenox
2. Downloading a source tarball from https://seamplex.com/feenox/dist/src/

C.2.3.1 Git repository

The main Git repository is hosted on GitHub at https://github.com/seamplex/feenox. It is public so it can be cloned either through HTTPS or SSH without needing any particular credentials. It can also be forked freely. See the Programming Guide for details about pull requests and/or write access to the main repository.

Ideally, the `main` branch should have a usable snapshot. All other branches might contain code that might not compile or might not run or might not be tested. If you find a commit in the main branch that does not pass the tests, please report it in the issue tracker ASAP.

After cloning the repository

```
git clone https://github.com/seamplex/feenox
```

the `autogen.sh` script has to be called to bootstrap the working tree, since the `configure` script is not stored in the repository but created from `configure.ac` (which is in the repository) by `autogen`.

Similarly, after updating the working tree with

```
git pull
```

it is recommended to re-run the `autogen.sh` script. It will do a `make clean` and re-compute the version string.

C.2.3.2 Source tarballs

When downloading a source tarball, there is no need to run `autogen.sh` since the `configure` script is already included in the tarball. This method cannot update the working tree. For each new FeenoX release, the whole tarball has to be downloaded again.

C.2.4 Configuration

To create a proper `Makefile` for the particular architecture, dependencies and compilation options, the script `configure` has to be executed. This procedure follows the GNU Coding Standards.

```
./configure
```

Without any particular options, `configure` will check if the mandatory GNU Scientific Library is available (both its headers and run-time library). If it is not, then the option `--enable-download-gsl` can be used. This option will try to use `wget` (which should be installed) to download a source tarball, uncompress is, configure and compile it. If these steps are successful, this GSL will be statically linked into the resulting FeenoX executable. If there is no internet connection, the `configure` script will say that the download failed. In that case, get the indicated tarball file manually , copy it into the current directory and re-run `./configure`.

The script will also check for the availability of optional dependencies. At the end of the execution, a summary of what was found (or not) is printed in the standard output:

```
$ ./configure
[...]
## ---------------------- ##
## Summary of dependencies ##
## ---------------------- ##
  GNU Scientific Library  from system
  SUNDIALS IDA            yes
  PETSc                   yes /usr/lib/petsc
  SLEPc                   no
[...]
```

If for some reason one of the optional dependencies is available but FeenoX should not use it, then pass `--without-sundials`, `--without-petsc` and/or `--without-slepc` as arguments. For example

```
$ ./configure --without-sundials --without-petsc
[...]
## ---------------------- ##
## Summary of dependencies ##
## ---------------------- ##
  GNU Scientific Library  from system
  SUNDIALS                no
  PETSc                   no
  SLEPc                   no
[...]
```

If configure complains about contradicting values from the cached ones, run `autogen.sh` again before `configure` or uncompress the source tarball in a fresh location.

To see all the available options run

```
./configure --help
```

C.2.5 Source code compilation

After the successful execution of `configure`, a `Makefile` is created. To compile FeenoX, just execute

```
make
```

Compilation should take a dozen of seconds. It can be even sped up by using the `-j` option

```
make -j8
```

The binary executable will be located in the `src` directory but a copy will be made in the base directory as well. Test it by running without any arguments

```
$ ./feenox
FeenoX v0.1.24-g6cfe063
a free no-fee no-X uniX-like finite-element(ish) computational engineering tool
```

```
usage: ./feenox [options] inputfile [replacement arguments]

  -h, --help         display usage and commmand-line help and exit
  -v, --version      display brief version information and exit
  -V, --versions     display detailed version information
  -s, --sumarize     list all symbols in the input file and exit

Instructions will be read from standard input if ""- is passed as
inputfile, i.e.

    $ echo "PRINT 2+2" | feenox -
    4

Report bugs at https://github.com/seamplex/feenox or to jeremy@seamplex.com
Feenox home page: https://www.seamplex.com/feenox/
$
```

The `-v` (or `--version`) option shows the version and a copyright notice:

```
$ ./feenox -v
FeenoX v0.1.24-g6cfe063
a free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Copyright (C) 2009--2021 jeremy theler
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 `-V` (or `--versions`) option shows the dates of the last commits, the compiler options and the versions of the linked libraries:

```
$ ./feenox -V
FeenoX v0.1.24-g6cfe063
a free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Last commit date   : Sun Aug 29 11:34:04 2021 -0300
Build date         : Sun Aug 29 11:44:50 2021 -0300
Build architecture : linux-gnu x86_64
Compiler           : gcc (Ubuntu 10.3.0-1ubuntu1) 10.3.0
Compiler flags     : -O3
Builder            : gtheler@chalmers
GSL version        : 2.6
SUNDIALS version   : 4.1.0
PETSc version      : Petsc Release Version 3.14.5, Mar 03, 2021
PETSc arch         :
PETSc options      : --build=x86_64-linux-gnu --prefix=/usr --includedir=${prefix}/include --mandir=${prefix ↩
     }/share/man --infodir=${prefix}/share/info --sysconfdir=/etc --localstatedir=/var --with-option- ↩
     checking=0 --with-silent-rules=0 --libdir=${prefix}/lib/x86_64-linux-gnu --runstatedir=/run --with- ↩
     maintainer-mode=0 --with-dependency-tracking=0 --with-debugging=0 --shared-library-extension=_real -- ↩
     with-shared-libraries --with-pic=1 --with-cc=mpicc --with-cxx=mpicxx --with-fc=mpif90 --with-cxx- ↩
     dialect=C++11 --with-opencl=1 --with-blas-lib=-lblas --with-lapack-lib=-llapack --with-scalapack=1 -- ↩
```

```
     with-scalapack-lib=-lscalapack-openmpi --with-ptscotch=1 --with-ptscotch-include=/usr/include/scotch -- ↩
     with-ptscotch-lib="-lptesmumps -lptscotch -lptscotcherr" --with-fftw=1 --with-fftw-include="[]" --with- ↩
     fftw-lib="-lfftw3 -lfftw3_mpi" --with-superlu_dist=1 --with-superlu_dist-include=/usr/include/superlu- ↩
     dist --with-superlu_dist-lib=-lsuperlu_dist --with-hdf5-include=/usr/include/hdf5/openmpi --with-hdf5- ↩
     lib="-L/usr/lib/x86_64-linux-gnu/hdf5/openmpi -L/usr/lib/x86_64-linux-gnu/openmpi/lib -lhdf5 -lmpi" -- ↩
     CXX_LINKER_FLAGS=-Wl,--no-as-needed --with-hypre=1 --with-hypre-include=/usr/include/hypre --with-hypre ↩
     -lib=-lHYPRE_core --with-mumps=1 --with-mumps-include="[]" --with-mumps-lib="-ldmumps -lzmumps -lsmumps ↩
      -lcmumps -lmumps_common -lpord" --with-suitesparse=1 --with-suitesparse-include=/usr/include/ ↩
     suitesparse --with-suitesparse-lib="-lumfpack -lamd -lcholmod -lklu" --with-superlu=1 --with-superlu- ↩
     include=/usr/include/superlu --with-superlu-lib=-lsuperlu --prefix=/usr/lib/petscdir/petsc3.14/x86_64- ↩
     linux-gnu-real --PETSC_ARCH=x86_64-linux-gnu-real CFLAGS="-g -O2 -ffile-prefix-map=/build/petsc-pVufYp/ ↩
     petsc-3.14.5+dfsg1=. -flto=auto -ffat-lto-objects -fstack-protector-strong -Wformat -Werror=format- ↩
     security -fPIC" CXXFLAGS="-g -O2 -ffile-prefix-map=/build/petsc-pVufYp/petsc-3.14.5+dfsg1=. -flto=auto  ↩
     -ffat-lto-objects -fstack-protector-strong -Wformat -Werror=format-security -fPIC" FCFLAGS="-g -O2 - ↩
     ffile-prefix-map=/build/petsc-pVufYp/petsc-3.14.5+dfsg1=. -flto=auto -ffat-lto-objects -fstack- ↩
     protector-strong -fPIC -ffree-line-length-0" FFLAGS="-g -O2 -ffile-prefix-map=/build/petsc-pVufYp/petsc ↩
     -3.14.5+dfsg1=. -flto=auto -ffat-lto-objects -fstack-protector-strong -fPIC -ffree-line-length-0"  ↩
     CPPFLAGS="-Wdate-time -D_FORTIFY_SOURCE=2" LDFLAGS="-Wl,-Bsymbolic-functions -flto=auto -Wl,-z,relro - ↩
     fPIC" MAKEFLAGS=w
SLEPc version      : SLEPc Release Version 3.14.2, Feb 01, 2021
$
```

C.2.6 Test suite

To be explained.

C.2.7 Installation

To be explained.

C.3 Advanced settings

C.3.1 Compiling with debug symbols

By default the C flags are `-O3`, without debugging. To add the `-g` flag, just use `CFLAGS` when configuring:

```
./configure CFLAGS="-g -O0"
```

C.3.2 Using a different compiler

Without PETSc, FeenoX uses the `CC` environment variable to set the compiler. So configure like

```
./configure CC=clang
```

When PETSc is detected FeenoX uses the `mpicc` executable, which is a wrapper to an actual C compiler with extra flags needed to find the headers and the MPI library. To change the wrapped compiler, you should set `MPICH_CC` or `OMPI_CC`, depending if you are using MPICH or OpenMPI. For example, to force MPICH to use `clang` do

```
./configure MPICH_CC=clang CC=clang
```

To know which is the default MPI implementation, just run `configure` without arguments and pay attention to the "Compiler" line in the "Summary of dependencies" section. For example, for OpenMPI a typical summary would be

```
## ----------------------- ##
## Summary of dependencies ##
## ----------------------- ##
  GNU Scientific Library  from system
  SUNDIALS                yes
  PETSc                   yes /usr/lib/petsc
  SLEPc                   yes /usr/lib/slepc
  Compiler                gcc -I/usr/lib/x86_64-linux-gnu/openmpi/include/openmpi -I/usr/lib/x86_64-linux- ↩
       gnu/openmpi/include -pthread -L/usr/lib/x86_64-linux-gnu/openmpi/lib -lmpi
```

For MPICH:

```
## ----------------------- ##
## Summary of dependencies ##
## ----------------------- ##
  GNU Scientific Library  from system
  SUNDIALS                yes
  PETSc                   yes /home/gtheler/libs/petsc-3.15.0 arch-linux2-c-debug
  SLEPc                   yes /home/gtheler/libs/slepc-3.15.1
  Compiler                gcc -Wl,-z,relro -I/usr/include/x86_64-linux-gnu/mpich -L/usr/lib/x86_64-linux-gnu ↩
        -lmpich
```

Other non-free implementations like Intel MPI might work but were not tested. However, it should be noted that the MPI implementation used to compile FeenoX has to match the one used to compile PETSc. Therefore, if you compiled PETSc on your own, it is up to you to ensure MPI compatibility. If you are using PETSc as provided by your distribution's repositories, you will have to find out which one was used (it is usually OpenMPI) and use the same one when compiling FeenoX.

The FeenoX executable will show the configured compiler and flags when invoked with the `--versions` option:

```
$ feenox --versions
FeenoX v0.1.47-g868dbb7-dirty
a free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Last commit date   : Mon Sep 6 16:39:53 2021 -0300
Build date         : Tue Sep 07 14:29:42 2021 -0300
Build architecture : linux-gnu x86_64
Compiler           : gcc (Debian 10.2.1-6) 10.2.1 20210110
Compiler flags     : -O3
Builder            : gtheler@tom
GSL version        : 2.6
SUNDIALS version   : 5.7.0
PETSc version      : Petsc Release Version 3.15.0, Mar 30, 2021
PETSc arch         : arch-linux2-c-debug
```

```
PETSc options      : --download-eigen --download-hdf5 --download-hypre --download-metis --download-mumps -- ↩
     download-parmetis --download-pragmatic --download-scalapack --with-x=0
SLEPc version      : SLEPc Release Version 3.15.1, May 28, 2021
$
```

Note that the reported values are the ones used in `configure` and not in `make`. Thus, the recommended way to set flags is in `configure` and not in `make`.

C.3.3 Compiling PETSc

To be explained.