FeenoX for Hackers

2024-06-14

Contents

1 Why

Why is FeenoX different from other "similar" tools?

To better illustrate FeenoX's unfair advantage (in the entrepreneurial sense), let us first consider what the options are when we need to write a technical report, paper or document:

Feature	Microsoft Word	Google Docs	Markdown[1]	(La)TeX
Aesthetics	×	×	✓	✓
Convertibility (to other formats)	∼	∼	✓	∼
Traceability	×	∼	✓	✓
Mobile-friendliness	×	✓	✓	×
Collaborativeness	×	✓	✓	∼
Licensing/openness	×	×	✓	✓
Non-nerd friendliness	✓	✓	∼	×

After analyzing the pros and cons of each alternative, at some point it should be evident that Markdown (plus friends) gives the best trade off. We can then perform a similar analysis for the options available in order to solve an engineering problem casted as a partial differential equation, say by using a finite-element formulation:

Feature	Desktop GUIs	Web frontends	FeenoX[2]	Libraries
Flexibility	∼	×	✓	✓
Scalability	×	∼	✓	✓
Traceability	×	∼	✓	✓
Cloud-friendliness	×	✓	✓	✓
Collaborativeness	×	✓	✓	∼
Licensing/openness	✓/∼/×	×	✓	✓
Non-nerd friendliness	✓	✓	∼	×

Therefore, FeenoX is—in a certain sense—to desktop FEA programs like

- Code_Aster with Salome-Meca, or
- CalculiX with PrePoMax

and to libraries like

- MoFEM or
- Sparselizard

what Markdown is to Word and (La)TeX, respectively and *deliberately*.

Unlike these other FEA tools, FeenoX provides…

[1] Here "Markdown" means (Pandoc + Git + Github / Gitlab / Gitea)
[2] Here "FeenoX" means (FeenoX + Gmsh + Paraview + Git + Github / Gitlab / Gitea)

- a ready-to-run executable (which uses Autotools and friends to compile) that reads the problem to be solved from an input file at run time (i.e. it is a program not a library) designed an implemented following the Unix programming philosophy:

```
$ feenox
FeenoX v0.3.317-g893dcd9
a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool

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

  -h, --help         display options and detailed explanations of commmand-line usage
  -v, --version      display brief version information and exit
  -V, --versions     display detailed version information
  --pdes             list the types of PROBLEMs that FeenoX can solve, one per line
  --elements_info    output a document with information about the supported element types
  --linear           force FeenoX to solve the PDE problem as linear
  --non-linear       force FeenoX to solve the PDE problem as non-linear

Run with --help for further explanations.
$
```

- a parser for a syntactically-sugared self-explanatory ASCII file (passed as the first non-optional argument to the `feenox` executable) with keywords that completely define the problem without requiring further human actions. Since the there is no need to recompile the binary for each problem, this allows efficient cloud-first workflows using containerized images or even provisioning by downloading binary tarballs or `.deb` packages.

- a few supported `PROBLEM` types and a mechanism to allow hacker and academics to add new PDEs (as explained in the next bullet). This bullet is about the fact that a regular user wanting to solve heat conduction (even with multi-material non-uniform conductivities) just needs to do

```
PROBLEM thermal
```

  and does not need to know nor write the weak form of the Poisson equation in the input file, since the vast majority of users will not know what a weak form is (even though other "similar" tools ask their users for that).

- a Git repository with GPL sources (and FDL documentation) where contributions are welcome. In particular, each partial differential equation that FeenoX can solve correspondens to one of the subdirectories of `src/pdes` that provide C entry points that the main mathematical framework calls as function pointer to build the elemental objects. The `autogen.sh` step (prior to `./configure` and `make`) detects the directory structure and includes all the subdirectories it finds as available problem types. They can be queried at runtime with the `--pdes` option:

```
$ feenox --pdes
laplace
mechanical
modal
neutron_diffusion
neutron_sn
thermal
$
```

The decision of extensibility through compiled code is, as the choice of making FeenoX a program and not a library, a thoughtful one. See FeenoX for academics for more details about how the extensibility mechanism works.

- continuous integration (using Github actions), an issue tracker (using Github issues and a discussion page (using Github discussions)
- a mechanism to expand command-line arguments as literal strings in the input file so as to allow parametric (and/or optimization) loops. For instance, if an input file `print.fee` looks like

  ```
  PRINT 2*${1}
  ```

  then

  ```
  $ for i in $(seq 1 5); do feenox print.fee $i; done
  2
  4
  6
  8
  10
  $
  ```

- the possibility to provide the input from `stdin` (so as to use it as a Unix pipe) by passing `-` as the input file path:

  ```
  $ for i in $(seq 1 5); do echo "PRINT 2*\${1}" | feenox - $i; done
  2
  4
  6
  8
  10
  $
  ```

- flexibility to handle many workflows, including web-based interfaces and thin command-line clients.

The input file…

- has a one-to-one correspondence with the human description of the problem
- is Git-traceable (the mesh is defined in a separate file created by Gmsh, which may or may not be tracked)
- allows the user to enter algebraic expressions whenever a numerical value is needed (everything is an expression)
- understands definitions (nouns) and instructions (verbs). FeenoX has an actual instruction pointer that loops over the instruction set (there might even be conditional blocks).
- is simple for simple files (but might get more complicated for mor complex problems). Remember Alan Kay's quote: "simple things should be simple and complex things should be possible."

Following the Unix rule of silence, the output is 100% user-defined: if there are not explicit output instructions, FeenoX will not write anything. And probably nothing will be computed (because FeenoX is smart and will not compute things that are not actually needed).

2 How

Feenox is a computational tool designed to be run on Unix servers as a part of a cloud-first workflow, optionally involving MPI communication among different servers to hande arbitrarily-large problems:

Check out the section about invocation in the FeenoX manual.

It has been written in C and designed under the Unix programming philosophy as quoted by Eric Raymond. Following the rule of composition, when solving PDEs FeenoX works very much as a Unix pipe between a mesher (such as Gmsh) and a post-processing tool (such as Paraview):

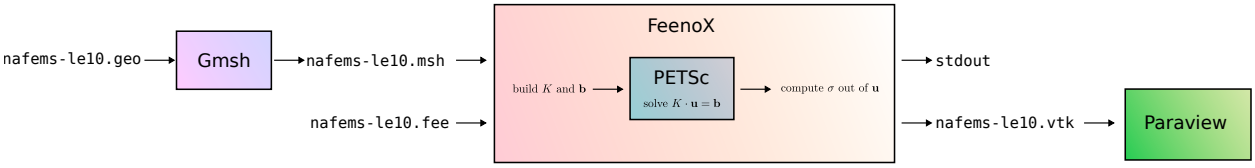
```
                            +------------+
mesh (*.msh)  }             |            |             { terminal
data (*.dat)  } input ----> |   FeenoX   |----> output { data files
input (*.fee) }             |            |             { post (vtk/msh)
                            +------------+
```

FeenoX consists of a binary executable which is compiled using GNU Autotools (i.e. `./autogen.sh && ./configure && make`) and uses three well-established and open source libraries:

a. The GNU Scientific Library for basic numerical computations
b. SUNDIALS IDA for solving systems of ODEs/DAEs
c. PETSc and SLEPc for solving PDEs

So even more, considering the NAFEMS LE10 Benchmark problem, it works as two "glue layers,"

1. between the mesher Gmsh and the PETSc library
2. between the PETSc library and the post-processor Paraview



The stock packages provided in most GNU/Linux distributions work perfectly well, but custom configured and compiled versions (e.g. with particular optimization flags or linked with non-standard MPI implementations) can be used as well.

An empty Debian-based GNU/Linux server (either `amd64` or `arm`) can be provisioned with a working FeenoX binary at `/usr/local/bin` ready to solve arbitrary problems by doing

```
sudo apt-get install -y libgsl-dev libsundials-dev petsc-dev slepc-dev
git clone https://github.com/seamplex/feenox
cd feenox
./autogen.sh
./configure
make
make install
```

> **Heads up!** If we wanted to be sure everything went smooth, we would need to take some time to install Gmsh and run the test suite:
>
> ```
> sudo apt-get install gmsh
> ```

```
make check
```

These steps are flexible enough so as to be integrated into containerization technologies (e.g. Docker files), continuous integration schemes (e.g. Github actions) or to suit any other particular needs (e.g. servers with custom PETSc installations or clusters multi-node MPI communication schemes). For instance, it is also possible to generate custom `.deb` (or `.rpm`) packages and make the server's `apt` manager to fetch and install them without needing to compile the source code at all.

Following the Unix rule of diversity, different compilers, both for the C code part of FeenoX as for the code in the dependencies (and their dependencies) can be used. So far there were tested

- GCC (free)
- Clang (free)
- Intel OneAPI (privative)

Also, different MPI implementations have been tested:

- OpenMPI (free, not to confuse with OpenMP)
- MPICH (free)
- Intel MPI (privative)

Feel free to raise any concerns you might have in our discussions forum.

3 What

FeenoX is a cloud-first back end for generic computational workflows to solve engineering-related problems:

- Basic mathematics
- Systems of ODEs/DAEs
- Laplace's equation
- Heat conduction
- Linear elasticity
- Modal analysis
- Neutron diffusion
- Neutron SN

3.1 Design

- FeenoX follows a fictitious (yet plausible) Software Design Requirements.
- The explanation of how FeenoX addresses the requirements can be found in the Software Design Specification.

3.2 Performance

- FeenoX's performance can be profiled and analyzed with the Google Benchmark library using this repository.
- A rough comparison of FeenoX's performance (and differences with respect to problem set up and execution) with respect to other similar tools can be found in this link: https://seamplex.com/feenox/tests/nafems/le10/

Check out FeenoX for Engineers and FeenoX for Academics for complementary information.