FeenoX for Academics

2025-06-17

Contents

1 What

FeenoX is a cloud-first Unix stand-alone **program** (i.e. something your run, not something you have to link existing code against) that reads an engineering-related problem in a plain-text file containing definitions and instructions. That is to say, it reads the problem to be solved at run time and does not require the user (most of the time these will be industry engineers and not hackers nor PhDs) to compile and link custom code in order to solve a problem because it is *not a library*. It does not require the users to write a weak form of the PDE they want to solve, because most of them will not even know what a weak form is (and they certainly do not need to know that). The user chooses from a set of built-in PDEs using the `PROBLEM` definition which internally resolves (at run time) a set of function pointers to the appropriate locations which will build the elemental objects which correspond the to chosen PDE. The list of available PDEs can be peeked by executing the `feenox` binary with the `--pdes` option:

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

During the compilation procedure (based on Autotools), the source tree in `src/pdes` is parsed. For each subdirectory, a new PDE is embedded into the compiled binary. See below for further details about this extensibility mechanism.

This program then solves the problem and, eventually, writes the outputs which the input file requests with explicit instructions (and nothing if nothing is asked for) and returns back to the Unix shell:

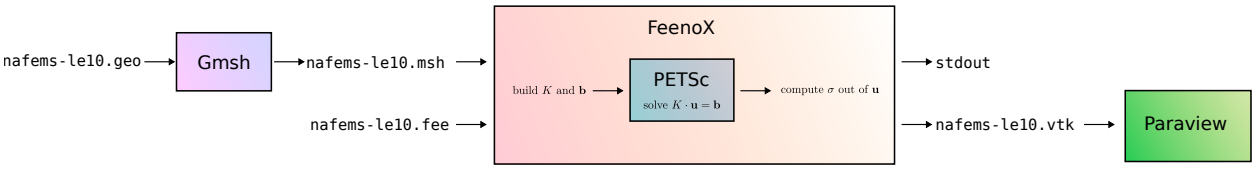
```
# NAFEMS Benchmark LE-10: thick plate pressure
PROBLEM mechanical DIMENSIONS 3
READ_MESH nafems-le10.msh    # mesh in millimeters
```

```
# LOADING: uniform normal pressure on the upper surface
BC upper    p=1       # 1 Mpa

# BOUNDARY CONDITIONS:
BC DCD'C'   v=0       # Face DCD'C' zero y-displacement
BC ABA'B'   u=0       # Face ABA'B' zero x-displacement
BC BCB'C'   u=0 v=0   # Face BCB'C' x and y displ. fixed
BC midplane w=0       #  z displacements fixed along mid-plane

# MATERIAL PROPERTIES: isotropic single-material properties
E = 210e3   # Young modulus in MPa
nu = 0.3    # Poisson's ratio

SOLVE_PROBLEM   # solve!

# print the direct stress y at D (and nothing more)
PRINT "σ_y @ D = " sigmay(2000,0,300) "MPa"
```

It can be seen as a Unix filter (or as a transfer function)

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

which, when zoomed in, acts as a "glue layer" between a mesher (Gmsh) and a library for solving large sparse problems (PETSc) which for linear elastic looks as follows:



Further discussion can be found in the tensile test tutorial. Check out the section about invocation in the FeenoX manual.

The design responds to a Software Requirements Specification document that acts as a "request for quotations" of a computational engineering tool that should satisfy some fictitious (but plausible) requirements. The Software Design Specification document explains how FeenoX addresses each requirement of the SRS.

In principle, even though FeenoX can solve generic numerical problems and systems of ordinary differential/algebraic equations, its main objective is to solve partial differential equations using the finite element method—eventually in parallel using the MPI standard. The current version can solve

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

Heads up! The background of FeenoX's main author is Nuclear Engineering. Hence,

- Two of the supported PDEs are related to neutron diffusion and transport.
- There is a PhD thesis (in Spanish only) that discusses the design and implementation of FeenoX in view of core-level neutronics.

As mentioned in the previous section, FeenoX provides a mechanism to add new types of PDEs by adding a new subdirectory to the `src/pdes` directory of the source tree and then re-bootstrapping, re-configuring and re-compiling the code.

Since in FeenoX's input file everything is an expression, the code is especially suited for verification using the method of manufactured solutions.

- See FeenoX for Engineers for links to further examples and tutorials.
- See FeenoX for Hackers for more details about the implementation and the code.

2 How

FeenoX tries to achieve its goals by…

- standing on both ethical (since it is free) and technical (since it is open source) ground while interacting with other free and open specimens such as
  - operating systems
  - libraries
  - compilers
  - pre and post-processing tools thus encouraging science and engineering to shift from privative environments into the free world.
- leveraging the Unix programming philosophy to come up with a cloud-first tool suitable to be automatically deployed and serve as the back end of web-based interfaces (fig. **??**).
- providing a ready-to-run program that reads an input file at run time (and not a library that has to be linked for each particular problem to be solved) as deliberate decision discussed in the Software Design Specification.
- designing and implementing an extensibility mechanism to allow hackers and/or academics to add new PDE formulations by adding a new subdirectory to `src/pdes` in the repository and then
  a. re-bootstrapping with `autogen.sh`,
  b. re-configuring with `configure`, and
  c. re-compiling with `make`

In effect, FeenoX provides a general mathematical framework to solve PDEs with a bunch of entry points (as C functions) where new types of PDEs (e.g. electromagnetism, fluid mechanics, etc.) can be added to the set of what FeenoX can solve. This general framework provides means to

- parse the input file, handle command-line arguments, read mesh files, assign variables, evaluate conditionals, write results, etc.

```
PROBLEM laplace 2D
READ_MESH square-$1.msh
[...]
WRITE_RESULTS FORMAT vtk
```

- handle material properties given as algebraic expressions involving pointwise-defined functions of space, temperature, time, etc.

```
MATERIAL steel     E=210e3*(1-1e-3*(T(x,y,z)-20))   nu=0.3
MATERIAL aluminum  E=69e3                           nu=7/25
```

- read problem-specific boundary conditions as algebraic expressions

```
sigma = 5.670374419e-8  # W m^2 / K^4 as in wikipedia
e = 0.98      # non-dimensional
T0 = 1000     # K
Tinf = 300    # K

BC left  T=T0
BC right q=sigma*e*(Tinf^4-T(x,y,z)^4)
```

- access shape functions and its derivatives evaluated either at Gauss points or at arbitrary locations for computing elementary contributions to

  - stiffness matrix
  - mass matrix
  - right-hand side vector

  For example, this snippet would build the elemental stiffness matrix for the Laplace problem:

```c
int build_laplace_Ki(element_t *e, unsigned int q) {
  double wdet = feenox_fem_compute_w_det_at_gauss(e, q);
  gsl_matrix *B = feenox_fem_compute_B_at_gauss(e, q);
  feenox_call(feenox_blas_BtB_accum(B, wdet, feenox.fem.Ki));
  return FEENOX_OK;
}
```

  The calls for computing the weights and the matrices with the shape functions and/or their derivatives currently support first and second-order iso-geometric elements, but other element types can be added as well.
  More complex cases involving non-uniform material properties, volumetric sources, etc. can be found in the actual source.

- solve the discretized equations using the appropriate PETSc/SLEPc objects, i.e.

  - KSP for linear static problems
  - SNES for non-linear static problems
  - TS for transient problems
  - EPS for eigenvalue problems

The particular functions that implement each problem type are located in subdirectories `src/pdes`, namely

- `laplace`
- `thermal`
- `mechanical`
- `modal`
- `neutron_diffusion`
- `neutron_sn`

Researchers with both knowledge of mathematical theory of finite elements and programming skills might, with the aid of the community, add support for other PDES. They might do that by using one of these directories (say `laplace`) as a template and

1. replace every occurrence of `laplace` in symbol names with the name of the new PDE
2. modify the initialization functions in `init.c` and set
   - the names of the unknowns
   - the names of the materials
   - the mathematical type and properties of problem
   - etc.
3. modify the contents of the elemental matrices in `bulk.c` in the FEM formulation of the problem being added
4. modify the contents of how the boundary conditions are parsed and set in `bc.c`
5. re-run `autogen.sh`, `./configure` and `make` to get a FeenoX executable with support for the new PDE.

As we mentioned in FeenoX for hackers, Alan Kay's says: "simple things should be simple and complex things should be possible." Of course, the addition of non-trivial PDEs is not straightforward, but possible (at that time we were discussing the first half of the quote, now we refer to the second part). The programming guide contains further details about how to contribute to the code base.

- See FeenoX for engineers for more details about the problem types FeenoX can solve and how to solve them.
- See FeenoX for hackers for more technical details about how FeenoX works.

3 Why

The world is already full of finite-element programs, and every day a grad student creates a new one from scratch. So why adding FeenoX to the already-crowded space of FEA tools? Because there are either

a. libraries which need code to use them such as
   - Sparselizard
   - MoFEM
   - FEniCS
   - MFEM

b. end-user programs which need a GUI such as
   - CalculiX
   - CodeAster

FeenoX sits in the middle. It is the only free and open-source tool that satisfies the Software Requirements Specification, including that…

- in order to solve a problem one needs to prepare an input file (not a script) which is read at run-time (not code which calls a library)
- these input files can expand generic command-line options using Bash syntax as `$1`, `$2`, etc., which allow parametric or optimization loops driven by higher-level scripts
- for solving PDEs, the input file has to refer to at least one Gmsh `.msh` file that defines the domain where the PDE is solved
- the material properties and boundary conditions are defined using physical groups and not individual nodes nor elements, so the input file is independent of the mesh and thus can be tracked with Git to increase traceability and repeatability.

- it uses the Unix philosophy which, among others, separates policy from mechanism and thus FeenoX is a natural choice for web-based interfaces like CAEplex.

See FeenoX for hackers for another explanation about why FeenoX is different from other computational tools.