FeenoX description

A cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool,

Jeremy Theler

Table of Contents

FeenoX description

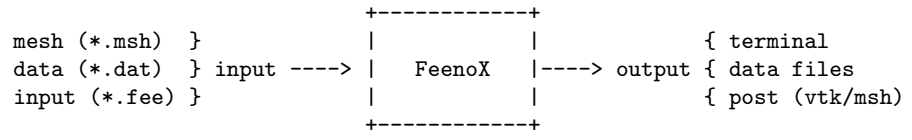
1 Overview

FeenoX is a computational tool that can solve engineering problems which are usually casted as differential-algebraic equations (DAEs) or partial differential equations (PDEs). It is to finite elements programs and libraries what Markdown is to Word and TeX, respectively. In particular, it can solve

- dynamical systems defined by a set of user-provided DAEs (such as plant control dynamics for example)
- mechanical elasticity
- heat conduction
- structural modal analysis
- neutron diffusion
- neutron transport

FeenoX reads a plain-text input file which contains the problem definition and writes 100%-user defined results in ASCII (through `PRINT` or other user-defined output instructions within the input file). For PDE problems, it needs a reference to at least one Gmsh (`http://gmsh.info/`) mesh file for the discretization of the domain. It can write post-processing views in either `.msh`, `.vtu` or `.vtk` formats.

Keep in mind that FeenoX is just a back end reading a set of input files and writing a set of output files following the design philosophy of Unix (separation, composition, representation, economy, extensibility, etc). Think of it as a transfer function (or a filter in computer-science jargon) between input files and output files:



Following the Unix programming philosophy, there are no graphical interfaces attached to the FeenoX core, although a wide variety of pre and post-processors can be used with FeenoX. To illustrate the transfer-function approach, consider the following input file that solves Laplace's equation $\nabla^2\phi = 0$ on a square with some space-dependent boundary conditions:

$\phi(x,y) = +y \quad$ for $x = -1$ (left)

$\phi(x,y) = -y \quad$ for $x = +1$ (right)

$\nabla\phi \cdot \hat{n} = \sin(\pi/2x) \quad$ for $y = -1$ (bottom)

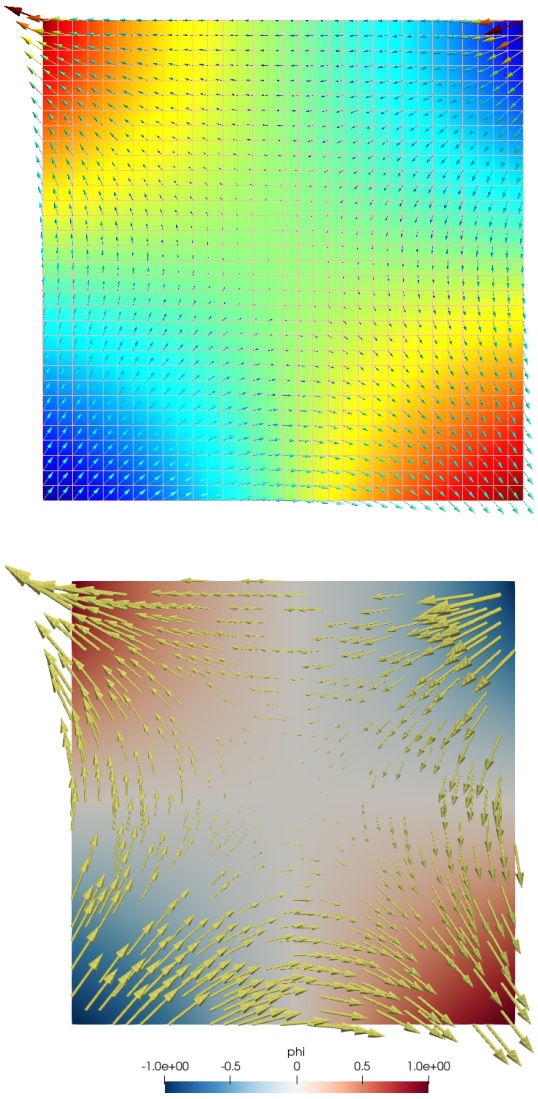
$\nabla\phi \cdot \hat{n} = 0 \quad$ for $y = +1$ (top)

```
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
```



Laplace's equation solved with FeenoX The `.msh` file can be post-processed with Gmsh (`http://gmsh.info/`), and the `.vtu`/`.vtk` file can be post-processed with Paraview (`https://www.paraview.org/`). See `https://www.caeplex.com` for a mobile-friendly web-based interface for solving finite elements in the cloud directly from the browser.

2 Introduction

FeenoX can be seen either as

- a syntactically-sweetened way of asking the computer to solve engineering-related mathematical problems, and/or
- a finite-element(ish) tool with a particular design basis.

Note that some of the problems solved with FeenoX might not actually rely on the finite element method, but on general mathematical models and even on the finite volumes method. That is why we say it is a finite-element(ish) tool.

In other words, FeenoX is a computational tool to solve

- dynamical systems written as sets of ODEs/DAEs (`https://seamplex.com/feenox/examples/daes.html`), or
- steady or transient heat conduction problems (`https://seamplex.com/feenox/examples/thermal.html`), or
- steady or quasi-static thermo-mechanical problems (`https://seamplex.com/feenox/examples/mechanical.html`), or
- modal analysis problems (`https://seamplex.com/feenox/examples/modal.html`), or
- core-level steady-state neutronics (`https://seamplex.com/feenox/examples/neutron_diffusion.html`), or
- community-contributed problems

in such a way that the input is a near-English text file that defines the problem to be solved.

One of the main features of this allegedly particular design basis is that **simple problems ought to have simple inputs** (*rule of simplicity*) or, quoting Alan Kay, "simple things should be simple, complex things should be possible."

For instance, to solve one-dimensional heat conduction over the domain $x \in [0,1]$ (which is indeed one of the most simple engineering problems we can find) the following input file is enough:

```
PROBLEM thermal 1D              # tell FeenoX what we want to solve
READ_MESH slab.msh              # read mesh in Gmsh's v4.1 format
k = 1                           # set uniform conductivity
BC left  T=0                    # set fixed temperatures as BCs
BC right T=1                    # "left" and "right" are defined in the mesh
SOLVE_PROBLEM                   # tell FeenoX we are ready to solve the problem
PRINT T(0.5)                    # ask for the temperature at x=0.5
```

```
$ feenox thermal-1d-dirichlet-constant-k.fee
0.5
$
```

The mesh is assumed to have been already created with Gmsh (`http://gmsh.info/`) (or any other pre-processing tool and converted to `.msh` format with Meshio (`https://github.com/nschloe/meshio`) for example). This assumption follows the *rule of composition* and prevents the actual input file from being polluted with mesh-dependent data (such as node coordinates and/or nodal loads) so as to keep it simple and make it Git (`https://git-scm.com/`)-friendly (*rule of generation*). The only link between the mesh and the FeenoX input

file is through physical groups (in the case above `left` and `right`) used to set boundary conditions and/or material properties.

Another design-basis decision is that **similar problems ought to have similar inputs** (*rule of least surprise*). So in order to have a space-dependent conductivity, we only have to replace one line in the input above: instead of defining a scalar k we define a function of x (we also update the output to show the analytical solution as well):

```
PROBLEM thermal 1D
READ_MESH slab.msh
k(x) = 1+x                      # space-dependent conductivity
BC left  T=0
BC right T=1
SOLVE_PROBLEM
PRINT T(1/2) log(1+1/2)/log(2)   # print numerical and analytical solutions
```

```
$ feenox thermal-1d-dirichlet-space-k.fee
0.584959  0.584963
$
```

The other main decision in FeenoX design is an **everything is an expression** design principle, meaning that any numerical input can be an algebraic expression (e.g. `T(1/2)` is the same as `T(0.5)`). If we want to have a temperature-dependent conductivity (which renders the problem non-linear) we can take advantage of the fact that $T(x)$ is available not only as an argument to `PRINT` but also for the definition of algebraic functions:

```
PROBLEM thermal 1D
READ_MESH slab.msh
k(x) = 1+T(x)                   # temperature-dependent conductivity
BC left  T=0
BC right T=1
SOLVE_PROBLEM
PRINT T(1/2) sqrt(1+(3*0.5))-1   # print numerical and analytical solutions
```

```
$ feenox thermal-1d-dirichlet-temperature-k.fee
0.581139  0.581139
$
```

Let us consider the famous chaotic Lorenz's dynamical system (`https://www.seamplex.com/feenox/examples/daes.html#lorenz-attractorthe-one-with-the-butterfly`). Here is one way of getting an image of the butterfly-shaped attractor using FeenoX to compute it and Gnuplot (`http://www.gnuplot.info/`) to draw it. Solve

$\dot{x} = \sigma \cdot (y - x)$
$\dot{y} = x \cdot (r - z) - y$
$\dot{z} = xy - bz$ for $0 < t < 40$ with initial conditions

$x(0) = -11$
$y(0) = -16$
$z(0) = 22.5$ 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 (`http://journals.ametsoc.org/doi/abs/10.1175/1520-0469%281963%29020%3C0130%3ADNF%3E2.0.CO%3B2`).

The following ASCII input file resembles the parameters, initial conditions and differential equations of the problem as naturally as possible:

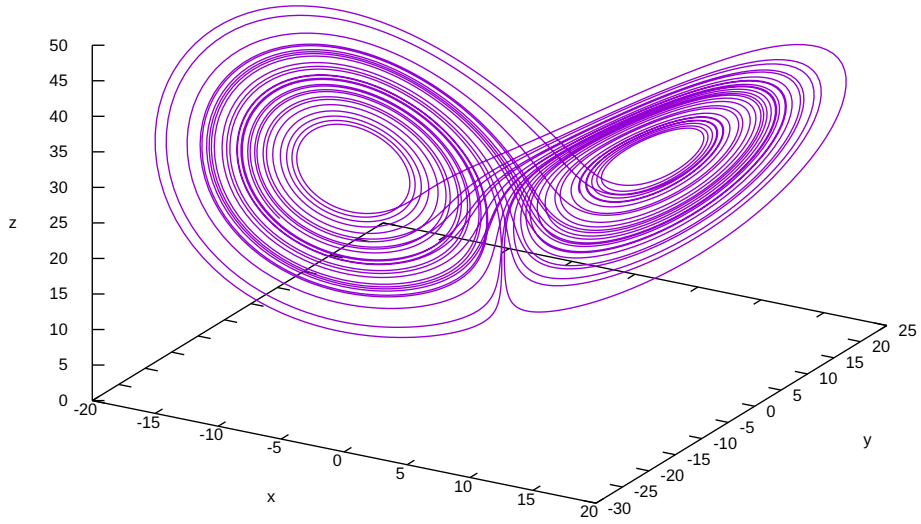
```
PHASE_SPACE x y z     # Lorenz attractor's phase space is x-y-z
end_time = 40         # we go from t=0 to 40 non-dimensional units
```

```
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
```

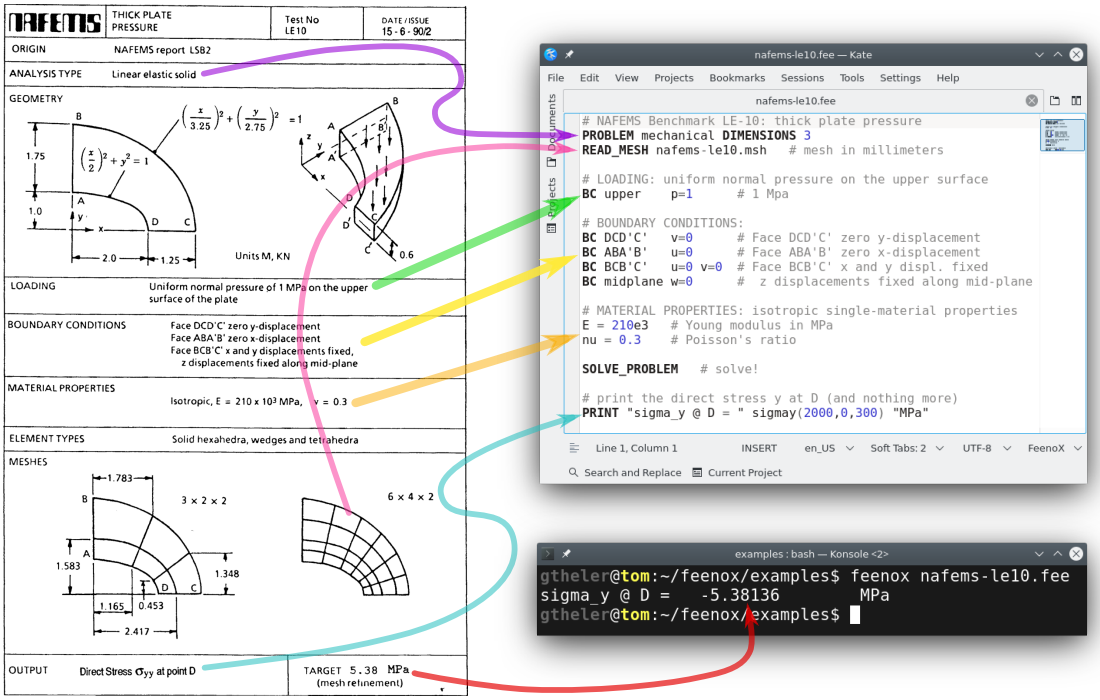


The Lorenz attractor solved with FeenoX and drawn with Gnuplot

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 (`(http://www.gnuplot.info/)`). Note the importance of relying on plain ASCII text formats both for input and output, as recommended by the Unix philosophy and the *rule of composition*: other programs can easily create inputs for FeenoX and other programs can easily understand FeenoX's outputs. This is essentially how Unix filters and pipes work.

Note the one-to-one correspondence between the human-friendly differential equations (written in TeX and rendered as typesetted mathematical symbols) and the computer-friendly input file that FeenoX reads.

Let us solve the linear elasticity benchmark problem NAFEMS LE10 "Thick plate pressure" (`https://www.seamplex.com/feenox/examples/mechanical.html#nafems-le10-thick-plate-pressure-benchmark`) with FeenoX. Note the one-to-one correspondence between the human-friendly problem statement from @fig:nafems-le10-problem-input and the FeenoX input file:



The NAFEMS LE10 problem statement and the corresponding FeenoX input

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

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

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

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

SOLVE_PROBLEM   # solve!

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

Here, "one-to-one" means that the input file does not need any extra definition which is not part of the problem formulation. Of course the cognizant engineer *can* give further definitions such as

- the linear solver and pre-conditioner
- the tolerances for iterative solvers

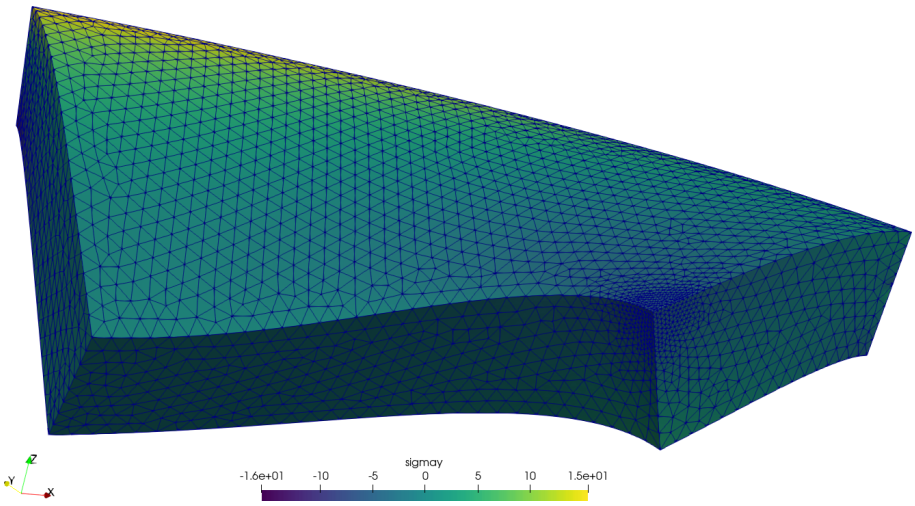
- options for computing stresses out of displacements
- etc.

However, she *is not obliged to* as–at least for simple problems—the defaults are reasonable. This is akin to writing a text in Markdown where one does not need to care if the page is A4 or letter (as, in most cases, the output will not be printed but rendered in a web browser).

The problem asks for the normal stress in the y direction σ_y at point "D," which is what FeenoX writes (and nothing else, *rule of economy*):

```
$ feenox nafems-le10.fee
sigma_y @ D =   -5.38016        MPa
$
```

Also note that since there is only one material, there is no need to do an explicit link between material properties and physical volumes in the mesh (*rule of simplicity*). And since the properties are uniform and isotropic, a single global scalar for E and a global single scalar for ν are enough.

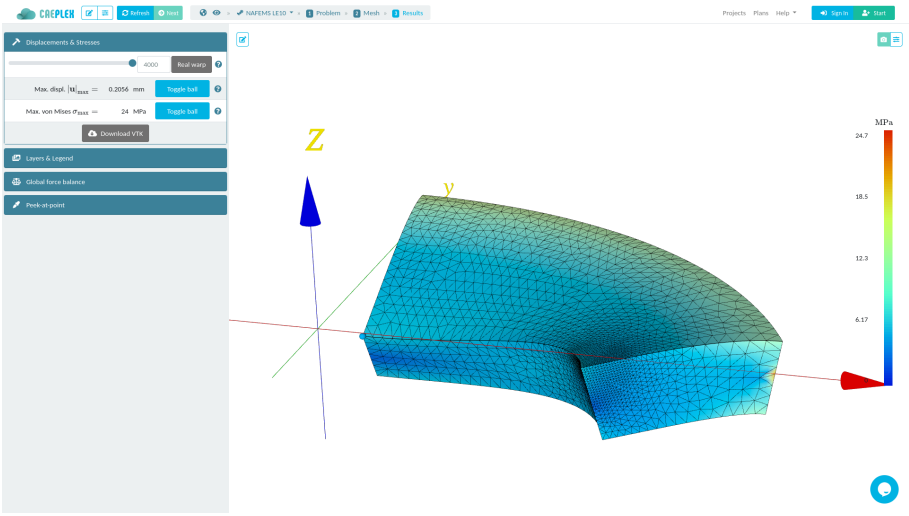


Normal stress σ_y refined around point D over 5,000x-warped displacements for LE10 created with Paraview

For the sake of visual completeness, post-processing data with the scalar distribution of σ_y and the vector field of displacements $[u, v, w]$ can be created by adding one line to the input file:

```
WRITE_MESH nafems-le10.vtk sigmay VECTOR u v w
```

This VTK file can then be post-processed to create interactive 3D views, still screenshots, browser and mobile-friendly webGL models, etc. In particular, using Paraview (`https://www.paraview.org`) one can get a colorful bitmapped PNG (the displacements are far more interesting than the stresses in this problem).



See also `https://caeplex.com/r/f1a82f` to see this very same LE10 problem solved in the mobile-friendly web-based interface CAEplex that uses FeenoX as the back end

Please note the following two points about both cases above:

1. The input files are very similar to the statements of each problem in plain English words (*rule of clarity*). Those with some experience may want to compare them to the inputs decks (sic) needed for other common FEA programs.
2. By design, 100% of FeenoX's output is controlled by the user. Had there not been any `PRINT` or `WRITE_MESH` instructions, the output would have been empty, following the *rule of silence*. This is a significant change with respect to traditional engineering codes that date back from times when one CPU hour was worth dozens (or even hundreds) of engineering hours. At that time, cognizant engineers had to dig into thousands of lines of data to search for a single individual result. Nowadays, following the *rule of economy*, it is actually far easier to ask the code to write only what is needed in the particular format that suits the user.

Some basic rules are

- FeenoX is just a **solver** working as a *transfer function* between input and output files.

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

  Following the *rules of separation, parsimony and diversity*, **there is no embedded graphical interface** but means of using generic pre and post processing tools—in particular, Gmsh (`http://gmsh.info/`) and Paraview (`https://www.paraview.org/`) respectively. See also CAEplex (`www.caeplex.com`) for a web-based interface.
- The input files should be syntactically sugared (`https://en.wikipedia.org/wiki/Syntactic_sugar`) so as to be as self-describing as possible.
- **Simple** problems ought to need **simple** input files.

- Similar problems ought to need similar input files.
- **Everything is an expression**. Whenever a number is expected, an algebraic expression can be entered as well. Variables, vectors, matrices and functions are supported. Here is how to replace the boundary condition on the right side of the slab above with a radiation condition:

```
sigma = 1       # non-dimensional stefan-boltzmann constant
e = 0.8         # emissivity
Tinf=1          # non-dimensional reference temperature
BC right q=sigma*e*(Tinf^4-T(x)^4)
```

  This "everything is an expression" principle directly allows the application of the Method of Manufactured Solutions for code verification.
- FeenoX should run natively in the cloud and be able to massively scale in parallel. See the Software Requirements Specification (`doc/sds.md`) and the Software Development Specification (`doc/sds.md`) for details.

Since it is free (as in freedom (`https://www.gnu.org/philosophy/free-sw.en.html`)) and open source, contributions to add features (and to fix bugs) are welcome. In particular, each kind of problem supported by FeenoX (thermal, mechanical, modal, etc.) has a subdirectory of source files which can be used as a template to add new problems, as implied in the "community-contributed problems" bullet above (*rules of modularity and extensibility*). See the documentation (`doc`) for details about how to contribute.

3 Running `feenox`

3.1 Invocation

The format for running the `feenox` program is:

```
feenox [options] inputfile [optional_extra_arguments] ...
```

The `feenox` executable supports the following options:

```
feenox [options] inputfile [replacement arguments] [petsc options]
```

`-h`, `--help`
: display options and detailed explanations of command-line usage

`-v`, `--version`
: display brief version information and exit

`-V`, `--versions`
: display detailed version information

`-c`, `--check`
: validates if the input file is sane or not

`--pdes`
: list the types of `PROBLEM`s that FeenoX can solve, one per line

`--elements_info`
: output a document with information about the supported element types

`--ast`
: dump an abstract syntax tree of the input

`--linear`
: force FeenoX to solve the PDE problem as linear

`--non-linear`
: force FeenoX to solve the PDE problem as non-linear

`--progress`
: print ASCII progress bars when solving PDEs

`--mumps`
: ask PETSc to use the direct linear solver MUMPS

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

```
$ echo 'PRINT 2+2' | feenox -
4
```

The optional `[replacement arguments]` part of the command line mean that each argument after the input file that does not start with an hyphen will be expanded verbatim in the input file in each occurrence of `$1`, `$2`, etc. For example

```
$ echo 'PRINT $1+$2' | feenox - 3 4
7
```

PETSc and SLEPc options can be passed in [petsc options] (or [options]) as well, with the difference that two hyphens have to be used instead of only once. For example, to pass the PETSc option -ksp_view the actual FeenoX invocation should be

```
$ feenox input.fee --ksp_view
```

For PETSc options that take values, en equal sign has to be used:

```
$ feenox input.fee --mg_levels_pc_type=sor
```

See https://www.seamplex.com/feenox/examples for annotated examples.

3.2 Compilation

These detailed compilation instructions are aimed at amd64 Debian-based GNU/Linux distributions. The compilation procedure follows the POSIX standard (https://en.wikipedia.org/wiki/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 (https://www.cygwin.com/), the packages should be installed through the Cygwin installer. WSL was not tested, but should work as well.

3.2.1 Quickstart

Note that the quickest way to get started is to download (https://www.seamplex.com/feenox/#download) an already-compiled statically-linked binary executable. Note that getting a binary is the quickest and easiest way to go but it is the less flexible one. Mind the following instructions if a binary-only option is not suitable for your workflow and/or you do need to compile the source code from scratch.

On a GNU/Linux box (preferably Debian-based), follow these quick steps. See @sec:details for the actual detailed explanations.

The Git repository has the latest sources repository. To compile, proceed as follows. If something goes wrong and you get an error, do not hesitate to ask in FeenoX's discussion page (https://github.com/seamplex/feenox/discussions).

> If you do not have Git or Autotools, download a source tarball (https://seamplex.com/feenox/dist/src/) and proceed with the usual configure & make procedure. See these instructions (doc/source.md).

1. Install mandatory dependencies

   ```
   sudo apt-get update
   sudo apt-get install git build-essential make 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. Bootstrap, 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. See the detailed compilation instructions (`compilation.md`) for an explanation.

5. Run test suite (optional)

   ```
   make check
   ```

6. Install the binary system wide (optional)

   ```
   sudo make install
   ```

   If you do not have root permissions, configure with your home directory as prefix and then make install as a regular user:

   ```
   ./configure --prefix=$HOME
   make
   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
```

3.2.2 Detailed configuration and compilation

The main target and development environment is Debian GNU/Linux (`https://www.debian.org/`), although it should be possible to compile FeenoX in any free GNU/Linux variant (and even the in non-free MacOS and/or Windows platforms) running in virtually any hardware platform. FeenoX can run be run either in HPC cloud servers or a Raspberry Pi, and almost everything that sits in the middle.

Following the Unix philosophy discussed in the SDS (`SDS.md`), FeenoX re-uses a lot of already-existing high-quality free and open source libraries that implement a wide variety of mathematical operations. This leads to a number of dependencies that FeenoX needs in order to implement certain features.

There is only one dependency that is mandatory, namely GNU GSL (`https://www.gnu.org/software/gsl/`) (see @sec:gsl), which if it not found then FeenoX cannot be compiled. All other dependencies are optional, meaning that FeenoX can be compiled but its capabilities will be partially reduced.

As per the SRS (`SRS.md`), all dependencies have to be available on mainstream GNU/Linux distributions and have to be free and open source software. 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).

In particular, PETSc (`https://petsc.org/release/`) (and SLEPc (`https://slepc.upv.es/`)) also depend on other mathematical libraries to perform particular operations such as low-level linear algebra operations. These extra dependencies can be either free

(such as LAPACK (http://www.netlib.org/lapack/)) or non-free (such as Intel's MKL (https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html)), 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.

3.2.2.1 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` or Intel's `icc` and the newer `icx` can also be used.

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 and its dependencies), but not to compile FeenoX if all the dependencies are installed from the operating system's 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 (https://github.com/seamplex/feenox/) then not only is `git` 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. The GNU Scientific Library The only run-time dependency is GNU GSL (https://www.gnu.org/software/gsl/) (not to be confused with Microsoft GSL (https://github.com/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 (https://www.gnu.org/software/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.

3.2.2.2 Optional dependencies

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

- SUNDIALS (https://computing.llnl.gov/projects/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 (https://petsc.org/) provides support for solving partial differential equations (PDEs). This dependency is needed when running inputs with the PROBLEM keyword.
- SLEPc (https://slepc.upv.es/) 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_sn.

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

- solve general mathematical problems such as the ones to compute the Fibonacci sequence (https://www.seamplex.com/feenox/examples/#the-fibonacci-sequence) or the Logistic map (https://www.seamplex.com/feenox/examples/#the-logistic-map),
- operate on functions, either algebraically or point-wise interpolated such as Computing the derivative of a function as a Unix filter (https://www.seamplex.com/feenox/examples/#computing-the-derivative-of-a-function-as-a-unix-filter)
- 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. When running the test suite (@sec:test-suite), those tests that need an optional dependency which was not found at compile time will be skipped. SUNDIALS SUNDIALS (https://computing.llnl.gov/projects/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 (https://www.seamplex.com/feenox/doc/feenox-manual.html#phase_space), like the Lorenz system (https://www.seamplex.com/feenox/examples/#lorenz-attractor-the-one-with-the-butterfly).

Install either by doing

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

or by following the instructions in the documentation. PETSc The Portable Extensible Toolkit for Scientific Computation ((https://petsc.org/)), pronounced PET-see (/pt-si/), 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 (https://www.seamplex.com/feenox/doc/feenox-manual.html#problem), like the NAFEMS LE10 benchmark problem (https://www.seamplex.com/feenox/examples/#nafems-le10-thick-plate-pressure-benchmark).

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 (https://petsc.org/release/install/) 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 and debugging 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).

SLEPc The Scalable Library for Eigenvalue Problem Computations (`https://slepc.upv.es/`), 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` (`https://www.seamplex.com/feenox/doc/feenox-manual.html#problem`) that need eigen-value computations, such as modal analysis of a cantilevered beam (`https://www.seamplex.com/feenox/examples/#five-natural-modes-of-a-cantilevered-wire`).

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.

3.2.2.3 FeenoX source code

There are two ways of getting FeenoX's 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/`

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 (`programming.md`) 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 can 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.sh`.

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. 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 source tarball has to be downloaded again.

3.2.2.4 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 (`https://www.gnu.org/prep/standards/`).

```
./configure
```

Without any particular options, `configure` will check if the mandatory GNU Scientific Library (`https://www.gnu.org/software/gsl/`) 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, 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` and/or clone/uncompress the source tarball in a fresh location.

To see all the available options run

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

3.2.2.5 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 v0.2.14-gbbf48c9
a free no-fee no-X uniX-like finite-element(ish) computational engineering tool

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

  -h, --help         display options and detailed explanations of command-line usage
  -v, --version      display brief version information and exit
  -V, --versions     display detailed version information

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

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

```
$ ./feenox -v
FeenoX v0.2.14-gbbf48c9
a free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Copyright © 2009--2022 https://seamplex.com/feenox
GNU General Public License v3+, https://www.gnu.org/licenses/gpl.html.
FeenoX is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
$
```

The `-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 version   : gcc (Debian 10.2.1-6) 10.2.1 20210110
Compiler expansion : gcc -Wl,-z,relro -I/usr/include/x86_64-linux-gnu/mpich -L/usr/lib/x86_64-linux-gnu -lmpich
Compiler flags     : -O3
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}/s
SLEPc version      : SLEPc Release Version 3.14.2, Feb 01, 2021
$
```

3.2.2.6 Test suite

The test (https://github.com/seamplex/feenox/tree/main/tests) directory contains a set of test cases whose output is known so that unintended regressions can be detected quickly (see the programming guide (programming.md) for more information). The test suite ought to be run after each modification in FeenoX's source code. It consists of a set of scripts and input files needed to solve dozens of cases. The output of each execution is compared to a reference solution. In case the output does not match the reference, the test suite fails.

After compiling FeenoX as explained in @sec:compilation, the test suite can be run with make check. Ideally everything should be green meaning the tests passed:

```
$ make check
Making check in src
make[1]: Entering directory '/home/gtheler/codigos/feenox/src'
make[1]: Nothing to be done for 'check'.
make[1]: Leaving directory '/home/gtheler/codigos/feenox/src'
make[1]: Entering directory '/home/gtheler/codigos/feenox'
cp -r src/feenox .
make  check-TESTS
make[2]: Entering directory '/home/gtheler/codigos/feenox'
make[3]: Entering directory '/home/gtheler/codigos/feenox'
XFAIL: tests/abort.sh
PASS: tests/algebraic_expr.sh
PASS: tests/beam-modal.sh
PASS: tests/beam-ortho.sh
PASS: tests/builtin.sh
PASS: tests/cylinder-traction-force.sh
PASS: tests/default_argument_value.sh
PASS: tests/expressions_constants.sh
PASS: tests/expressions_variables.sh
PASS: tests/expressions_functions.sh
PASS: tests/exp.sh
PASS: tests/i-beam-euler-bernoulli.sh
PASS: tests/iaea-pwr.sh
PASS: tests/iterative.sh
PASS: tests/fit.sh
PASS: tests/function_algebraic.sh
PASS: tests/function_data.sh
PASS: tests/function_file.sh
PASS: tests/function_vectors.sh
PASS: tests/integral.sh
PASS: tests/laplace2d.sh
PASS: tests/materials.sh
PASS: tests/mesh.sh
PASS: tests/moment-of-inertia.sh
PASS: tests/nafems-le1.sh
PASS: tests/nafems-le10.sh
PASS: tests/nafems-le11.sh
PASS: tests/nafems-t1-4.sh
PASS: tests/nafems-t2-3.sh
PASS: tests/neutron_diffusion_src.sh
PASS: tests/neutron_diffusion_keff.sh
PASS: tests/parallelepiped.sh
PASS: tests/point-kinetics.sh
PASS: tests/print.sh
PASS: tests/thermal-1d.sh
```

```
PASS: tests/thermal-2d.sh
PASS: tests/trig.sh
PASS: tests/two-cubes-isotropic.sh
PASS: tests/two-cubes-orthotropic.sh
PASS: tests/vector.sh
XFAIL: tests/xfail-few-properties-ortho-young.sh
XFAIL: tests/xfail-few-properties-ortho-poisson.sh
XFAIL: tests/xfail-few-properties-ortho-shear.sh
============================================================================
Testsuite summary for feenox v0.2.6-g3237ce9
============================================================================
# TOTAL: 43
# PASS:  39
# SKIP:  0
# XFAIL: 4
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
make[3]: Leaving directory '/home/gtheler/codigos/feenox'
make[2]: Leaving directory '/home/gtheler/codigos/feenox'
make[1]: Leaving directory '/home/gtheler/codigos/feenox'
$
```

The `XFAIL` result means that those cases are expected to fail (they are there to test if FeenoX can handle errors). Failure would mean they passed. In case FeenoX was not compiled with any optional dependency, the corresponding tests will be skipped. Skipped tests do not mean any failure, but that the compiled FeenoX executable does not have the full capabilities. For example, when configuring with `./configure --without-petsc` (but with SUNDIALS), the test suite output should be a mixture of green and blue:

```
$ ./configure --without-petsc
[...]
configure: creating ./src/version.h
## ----------------------- ##
## Summary of dependencies ##
## ----------------------- ##
  GNU Scientific Library  from system
  SUNDIALS                yes
  PETSc                   no
  SLEPc                   no
  Compiler                gcc
checking that generated files are newer than configure... done
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating doc/Makefile
config.status: executing depfiles commands
$ make
[...]
$ make check
Making check in src
make[1]: Entering directory '/home/gtheler/codigos/feenox/src'
make[1]: Nothing to be done for 'check'.
make[1]: Leaving directory '/home/gtheler/codigos/feenox/src'
make[1]: Entering directory '/home/gtheler/codigos/feenox'
cp -r src/feenox .
make  check-TESTS
```

```
make[2]: Entering directory '/home/gtheler/codigos/feenox'
make[3]: Entering directory '/home/gtheler/codigos/feenox'
XFAIL: tests/abort.sh
PASS: tests/algebraic_expr.sh
SKIP: tests/beam-modal.sh
SKIP: tests/beam-ortho.sh
PASS: tests/builtin.sh
SKIP: tests/cylinder-traction-force.sh
PASS: tests/default_argument_value.sh
PASS: tests/expressions_constants.sh
PASS: tests/expressions_variables.sh
PASS: tests/expressions_functions.sh
PASS: tests/exp.sh
SKIP: tests/i-beam-euler-bernoulli.sh
SKIP: tests/iaea-pwr.sh
PASS: tests/iterative.sh
PASS: tests/fit.sh
PASS: tests/function_algebraic.sh
PASS: tests/function_data.sh
PASS: tests/function_file.sh
PASS: tests/function_vectors.sh
PASS: tests/integral.sh
SKIP: tests/laplace2d.sh
PASS: tests/materials.sh
PASS: tests/mesh.sh
PASS: tests/moment-of-inertia.sh
SKIP: tests/nafems-le1.sh
SKIP: tests/nafems-le10.sh
SKIP: tests/nafems-le11.sh
SKIP: tests/nafems-t1-4.sh
SKIP: tests/nafems-t2-3.sh
SKIP: tests/neutron_diffusion_src.sh
SKIP: tests/neutron_diffusion_keff.sh
SKIP: tests/parallelepiped.sh
PASS: tests/point-kinetics.sh
PASS: tests/print.sh
SKIP: tests/thermal-1d.sh
SKIP: tests/thermal-2d.sh
PASS: tests/trig.sh
SKIP: tests/two-cubes-isotropic.sh
SKIP: tests/two-cubes-orthotropic.sh
PASS: tests/vector.sh
SKIP: tests/xfail-few-properties-ortho-young.sh
SKIP: tests/xfail-few-properties-ortho-poisson.sh
SKIP: tests/xfail-few-properties-ortho-shear.sh
============================================================================
Testsuite summary for feenox v0.2.6-g3237ce9
============================================================================
# TOTAL: 43
# PASS:  21
# SKIP:  21
# XFAIL: 1
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
make[3]: Leaving directory '/home/gtheler/codigos/feenox'
make[2]: Leaving directory '/home/gtheler/codigos/feenox'
```

```
make[1]: Leaving directory '/home/gtheler/codigos/feenox'
$
```

To illustrate how regressions can be detected, let us add a bug deliberately and re-run the test suite.

Edit the source file that contains the shape functions of the second-order tetrahedra `src/mesh/tet10.c`, find the function `feenox_mesh_tet10_h()` and randomly change a sign, i.e. replace

```
      return t*(2*t-1);
```

with

```
      return t*(2*t+1);
```

Save, recompile, and re-run the test suite to obtain some red:

```
$ git diff src/mesh/
diff --git a/src/mesh/tet10.c b/src/mesh/tet10.c
index 72bc838..293c290 100644
--- a/src/mesh/tet10.c
+++ b/src/mesh/tet10.c
@@ -227,7 +227,7 @@ double feenox_mesh_tet10_h(int j, double *vec_r) {
       return s*(2*s-1);
       break;
     case 3:
-      return t*(2*t-1);
+      return t*(2*t+1);
       break;

     case 4:
$ make
[...]
$ make check
Making check in src
make[1]: Entering directory '/home/gtheler/codigos/feenox/src'
make[1]: Nothing to be done for 'check'.
make[1]: Leaving directory '/home/gtheler/codigos/feenox/src'
make[1]: Entering directory '/home/gtheler/codigos/feenox'
cp -r src/feenox .
make  check-TESTS
make[2]: Entering directory '/home/gtheler/codigos/feenox'
make[3]: Entering directory '/home/gtheler/codigos/feenox'
XFAIL: tests/abort.sh
PASS: tests/algebraic_expr.sh
FAIL: tests/beam-modal.sh
PASS: tests/beam-ortho.sh
PASS: tests/builtin.sh
PASS: tests/cylinder-traction-force.sh
PASS: tests/default_argument_value.sh
PASS: tests/expressions_constants.sh
PASS: tests/expressions_variables.sh
PASS: tests/expressions_functions.sh
PASS: tests/exp.sh
PASS: tests/i-beam-euler-bernoulli.sh
PASS: tests/iaea-pwr.sh
PASS: tests/iterative.sh
PASS: tests/fit.sh
PASS: tests/function_algebraic.sh
PASS: tests/function_data.sh
PASS: tests/function_file.sh
```

```
PASS: tests/function_vectors.sh
PASS: tests/integral.sh
PASS: tests/laplace2d.sh
PASS: tests/materials.sh
PASS: tests/mesh.sh
PASS: tests/moment-of-inertia.sh
PASS: tests/nafems-le1.sh
FAIL: tests/nafems-le10.sh
FAIL: tests/nafems-le11.sh
PASS: tests/nafems-t1-4.sh
PASS: tests/nafems-t2-3.sh
PASS: tests/neutron_diffusion_src.sh
PASS: tests/neutron_diffusion_keff.sh
FAIL: tests/parallelepiped.sh
PASS: tests/point-kinetics.sh
PASS: tests/print.sh
PASS: tests/thermal-1d.sh
PASS: tests/thermal-2d.sh
PASS: tests/trig.sh
PASS: tests/two-cubes-isotropic.sh
PASS: tests/two-cubes-orthotropic.sh
PASS: tests/vector.sh
XFAIL: tests/xfail-few-properties-ortho-young.sh
XFAIL: tests/xfail-few-properties-ortho-poisson.sh
XFAIL: tests/xfail-few-properties-ortho-shear.sh
============================================================================
Testsuite summary for feenox v0.2.6-g3237ce9
============================================================================
# TOTAL: 43
# PASS:  35
# SKIP:  0
# XFAIL: 4
# FAIL:  4
# XPASS: 0
# ERROR: 0
============================================================================
See ./test-suite.log
Please report to jeremy@seamplex.com
============================================================================
make[3]: *** [Makefile:1152: test-suite.log] Error 1
make[3]: Leaving directory '/home/gtheler/codigos/feenox'
make[2]: *** [Makefile:1260: check-TESTS] Error 2
make[2]: Leaving directory '/home/gtheler/codigos/feenox'
make[1]: *** [Makefile:1791: check-am] Error 2
make[1]: Leaving directory '/home/gtheler/codigos/feenox'
make: *** [Makefile:1037: check-recursive] Error 1
$
```

3.2.2.7 Installation

To be able to execute FeenoX from any directory, the binary has to be copied to a directory available in the `PATH` environment variable. If you have root access, the easiest and cleanest way of doing this is by calling `make install` with `sudo` or `su`:

```
$ sudo make install
Making install in src
make[1]: Entering directory '/home/gtheler/codigos/feenox/src'
gmake[2]: Entering directory '/home/gtheler/codigos/feenox/src'
 /usr/bin/mkdir -p '/usr/local/bin'
```

```
  /usr/bin/install -c feenox '/usr/local/bin'
gmake[2]: Nothing to be done for 'install-data-am'.
gmake[2]: Leaving directory '/home/gtheler/codigos/feenox/src'
make[1]: Leaving directory '/home/gtheler/codigos/feenox/src'
make[1]: Entering directory '/home/gtheler/codigos/feenox'
cp -r src/feenox .
make[2]: Entering directory '/home/gtheler/codigos/feenox'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/gtheler/codigos/feenox'
make[1]: Leaving directory '/home/gtheler/codigos/feenox'
$
```

If you do not have root access or do not want to populate `/usr/local/bin`, you can either

- Configure with a different prefix (not covered here), or
- Copy (or symlink) the `feenox` executable to `$HOME/bin`:

  ```
  mkdir -p ${HOME}/bin
  cp feenox ${HOME}/bin
  ```

  If you plan to regularly update FeenoX (which you should), you might want to symlink instead of copy so you do not need to update the binary in `$HOME/bin` each time you recompile:

  ```
  mkdir -p ${HOME}/bin
  ln -sf feenox ${HOME}/bin
  ```

Check that FeenoX is now available from any directory (note the command is `feenox` and not `./feenox`):

```
$ cd
$ feenox -v
FeenoX v0.2.14-gbbf48c9
a free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Copyright © 2009--2022 https://seamplex.com/feenox
GNU General Public License v3+, https://www.gnu.org/licenses/gpl.html.
FeenoX is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
$
```

If it is not and you went through the `$HOME/bin` path, make sure it is in the `PATH` (pun). Add

```
export PATH=${PATH}:${HOME}/bin
```

to your `.bashrc` in your home directory and re-login.

3.2.3 Advanced settings

3.2.3.1 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"
```

3.2.3.2 Using a different compiler

FeenoX uses the `CC` environment variable to set the compiler. So configure like

```
export CC=clang; ./configure
```

Note that the `CC` variable has to be *exported* and not *passed* to configure. That is to say, don't configure like

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

Mind also the following environment variables when using MPI-enabled PETSc:

- `MPICH_CC`
- `OMPI_CC`
- `I_MPI_CC`

Depending on how your system is configured, this last command might show `clang` but not actually use it. The FeenoX executable will show the configured compiler and flags when invoked with the `--versions` option:

```
$ feenox --versions
FeenoX v0.2.14-gbbf48c9
a free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Last commit date   : Sat Feb 12 15:35:05 2022 -0300
Build date         : Sat Feb 12 15:35:44 2022 -0300
Build architecture : linux-gnu x86_64
Compiler version   : gcc (Debian 10.2.1-6) 10.2.1 20210110
Compiler expansion : gcc -Wl,-z,relro -I/usr/include/x86_64-linux-gnu/mpich -L/usr/lib/x86_64-linux-gnu -lmpich
Compiler flags     : -O3
Builder            : gtheler@tom
GSL version        : 2.6
SUNDIALS version   : 5.7.0
PETSc version      : Petsc Release Version 3.16.3, Jan 05, 2022
PETSc arch         : arch-linux-c-debug
PETSc options      : --download-eigen --download-hdf5 --download-hypre --download-metis --download-mumps --down
SLEPc version      : SLEPc Release Version 3.16.1, Nov 17, 2021
$
```

You can check which compiler was actually used by analyzing the `feenox` binary as

```
$ objdump -s --section .comment ./feenox

./feenox:     file format elf64-x86-64

Contents of section .comment:
 0000 4743433a 20284465 6269616e 2031322e  GCC: (Debian 12.
 0010 322e302d 31342920 31322e32 2e300044  2.0-14) 12.2.0.D
 0020 65626961 6e20636c 616e6720 76657273  ebian clang vers
 0030 696f6e20 31342e30 2e3600             ion 14.0.6.
$
```

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. FeenoX has been tested using PETSc compiled with

- MPICH
- OpenMPI
- Intel MPI

3.2.3.3 Compiling PETSc

Particular explanation for FeenoX is to be done. For now, follow the general explanation from PETSc's website (`https://petsc.org/release/install/`).

```
export PETSC_DIR=$PWD
export PETSC_ARCH=arch-linux-c-opt
./configure --with-debugging=0 --download-mumps --download-scalapack --with-cxx=0 --COPTFLAGS=-O3 --FOPTFLAGS=-
```

```
export PETSC_DIR=$PWD
./configure --with-debugging=0 --with-openmp=0 --with-x=0 --with-cxx=0 --COPTFLAGS=-O3 --FOPTFLAGS=-O3
make PETSC_DIR=/home/ubuntu/reflex-deps/petsc-3.17.2 PETSC_ARCH=arch-linux-c-opt all
```

4 Examples

See https://www.seamplex.com/feenox/examples for updated information.

- Basic mathematics
  - Hello World and Universe! (https://seamplex.com/feenox/examples/basic.html#hello-world-and-universe)
  - Ten ways of computing π (https://seamplex.com/feenox/examples/basic.html#ten-ways-of-computing-pi)
  - Financial decisions under inflation (https://seamplex.com/feenox/examples/basic.html#financial-decisions-under-inflation)
  - The logistic map (https://seamplex.com/feenox/examples/basic.html#the-logistic-map)
  - The Fibonacci sequence (https://seamplex.com/feenox/examples/basic.html#the-fibonacci-sequence)
    - Using the closed-form formula as a function (https://seamplex.com/feenox/examples/basic.html#using-the-closed-form-formula-as-a-function)
    - Using a vector (https://seamplex.com/feenox/examples/basic.html#using-a-vector)
    - Solving an iterative problem (https://seamplex.com/feenox/examples/basic.html#solving-an-iterative-problem)
  - Computing the derivative of a function as a Unix filter (https://seamplex.com/feenox/examples/basic.html#computing-the-derivative-of-a-function-as-a-unix-filter)
  - On the evaluation of thermal expansion coefficients (https://seamplex.com/feenox/examples/basic.html#on-the-evaluation-of-thermal-expansion-coefficients)
  - Buffon's needle (https://seamplex.com/feenox/examples/basic.html#buffons-needle)
- Ordinary Differential Equations & Differential-Algebraic Equations
  - Lorenz' attractor—the one with the butterfly (https://seamplex.com/feenox/examples/daes.html#lorenz-attractorthe-one-with-the-butterfly)
  - The double pendulum (https://seamplex.com/feenox/examples/daes.html#the-double-pendulum)
  - Vertical boiling channel (https://seamplex.com/feenox/examples/daes.html#vertical-boiling-channel)
    - Original Clausse-Lahey formulation with uniform power distribution (https://seamplex.com/feenox/examples/daes.html#original-clausse-lahey-formulation-with-uniform-power-distribution)
    - Arbitrary power distribution (https://seamplex.com/feenox/examples/daes.html#arbitrary-power-distribution)
  - Reactor point kinetics (https://seamplex.com/feenox/examples/daes.html#reactor-point-kinetics)
    - Cinética puntual directa con reactividad vs tiempo (https://seamplex.com/feenox/examples/daes.html#cinética-puntual-directa-con-reactividad-vs.-tiempo)

        - Cinética inversa (https://seamplex.com/feenox/examples/daes.html#cinética-inversa)
        - Control de inestabilidades de xenón (https://seamplex.com/feenox/examples/daes.html#control-de-inestabilidades-de-xenón)
        - Mapas de diseño (https://seamplex.com/feenox/examples/daes.html#mapas-de-diseño)
- Laplace's equation
    - How to solve a maze without AI (https://seamplex.com/feenox/examples/laplace.html#how-to-solve-a-maze-without-ai)
        - Transient top-down (https://seamplex.com/feenox/examples/laplace.html#transient-top-down)
        - Transient bottom-up (https://seamplex.com/feenox/examples/laplace.html#transient-bottom-up)
    - Potential flow around an airfoil profile (https://seamplex.com/feenox/examples/laplace.html#potential-flow-around-an-airfoil-profile)
- Heat conduction
    - Thermal slabs (https://seamplex.com/feenox/examples/thermal.html#thermal-slabs)
        - One-dimensional linear (https://seamplex.com/feenox/examples/thermal.html#one-dimensional-linear)
    - Transient heat conduction from steady-state by "turning off" BCs (https://seamplex.com/feenox/examples/thermal.html#transient-heat-conduction-from-steady-state-by-turning-off-bcs)
    - Non-dimensional transient heat conduction on a cylinder (https://seamplex.com/feenox/examples/thermal.html#non-dimensional-transient-heat-conduction-on-a-cyli
    - Non-dimensional transient heat conduction with time-dependent properties (https://seamplex.com/feenox/examples/thermal.html#non-dimensional-transient-heat-conduction-with-time-dependent-properties)
- Linear elasticity
    - NAFEMS LE10 "Thick plate pressure" benchmark (https://seamplex.com/feenox/examples/mechanical.html#nafems-le10-thick-plate-pressure-benchmark)
    - NAFEMS LE11 "Solid Cylinder/Taper/Sphere-Temperature" benchmark (https://seamplex.com/feenox/examples/mechanical.html#nafems-le11-solid-cylindertapersphere-temperature-benchmark)
    - NAFEMS LE1 "Elliptical membrane" plane-stress benchmark (https://seamplex.com/feenox/examples/mechanical.html#nafems-le1-elliptical-membrane-plane-stress-benchmark)
    - Parametric study on a cantilevered beam (https://seamplex.com/feenox/examples/mechanical.html#parametric-study-on-a-cantilevered-beam)
    - Parallelepiped whose Young's modulus is a function of the temperature (https://seamplex.com/feenox/examples/mechanical.html#parallelepiped-whose-youngs-modulus-is-a-function-of-the-temperature)

        - Thermal problem (https://seamplex.com/feenox/examples/mechanical.html#thermal-problem)
        - Mechanical problem (https://seamplex.com/feenox/examples/mechanical.html#mechanical-problem)
    - Orthotropic free expansion of a cube (https://seamplex.com/feenox/examples/mechanical.html#orthotropic-free-expansion-of-a-cube)
    - Thermo-elastic expansion of finite cylinders (https://seamplex.com/feenox/examples/mechanical.html#thermo-elastic-expansion-of-finite-cylinders)
    - Temperature-dependent material properties (https://seamplex.com/feenox/examples/mechanical.html#temperature-dependent-material-properties)
    - Two cubes compressing each other (https://seamplex.com/feenox/examples/mechanical.html#two-cubes-compressing-each-other)
- Mechanical modal analysis
    - Optimizing the length of a tuning fork (https://seamplex.com/feenox/examples/modal.html#optimizing-the-length-of-a-tuning-fork)
    - Five natural modes of a cantilevered wire (https://seamplex.com/feenox/examples/modal.html#five-natural-modes-of-a-cantilevered-wire)
- Neutron diffusion
    - IAEA 2D PWR Benchmark (https://seamplex.com/feenox/examples/neutron_diffusion.html#iaea-2d-pwr-benchmark)
    - IAEA 3D PWR Benchmark (https://seamplex.com/feenox/examples/neutron_diffusion.html#iaea-3d-pwr-benchmark)
    - Cube-spherical bare reactor (https://seamplex.com/feenox/examples/neutron_diffusion.html#cube-spherical-bare-reactor)
    - Illustration of the XS dilution & smearing effect (https://seamplex.com/feenox/examples/neutron_diffusion.html#illustration-of-the-xs-dilution-smearing-effect)
- Neutron transport using S_N
    - Reed's problem (https://seamplex.com/feenox/examples/neutron_sn.html#reeds-problem)
    - Azmy's problem (https://seamplex.com/feenox/examples/neutron_sn.html#azmys-problem)
        - Second-order complete structured rectangular grid (https://seamplex.com/feenox/examples/neutron_sn.html#second-order-complete-structured-rectangular-gri
        - Fist-order locally-refined unstructured triangular grid (https://seamplex.com/feenox/examples/neutron_sn.html#fist-order-locally-refined-unstructured-tria
        - Flux profiles with ray effect (https://seamplex.com/feenox/examples/neutron_sn.html#flux-profiles-with-ray-effect)

5 Tutorials

See `https://www.seamplex.com/feenox/doc/tutorials` for updated information.

1. Setting up your workspace (`https://www.seamplex.com/feenox/doc/tutorials/000-setup`)

5.1 General tutorials

1. Overview the tensile test case (`https://www.seamplex.com/feenox/doc/tutorials/110-tensile-test`)
2. Fun & games solving mazes with PDES instead of AI (`https://www.seamplex.com/feenox/doc/tutorials/120-mazes`)

5.2 Detailed functionality

1. Input files, expressions and command-line arguments
2. Static & transient cases
3. Functions & functionals
4. Vectors & matrices
5. Differential-algebraic equations
6. Meshes & distributions

5.3 Physics tutorials

1. The Laplace equation
2. Heat conduction (`https://www.seamplex.com/feenox/doc/tutorials/320-thermal`)
3. Linear elasticity
4. Modal analysis
5. Thermo-mechanical analysis
6. Neutron diffusion
7. Neutron transport

6 Description

FeenoX solves a problem defined in an plain-text input file and writes user-defined outputs to the standard output and/or files, either also plain-text or with a particular format for further post-processing. The syntax of this input file is designed to be as self-describing as possible, using English keywords that explains FeenoX what problem it has to solve in a way is understandable by both humans and computers. Keywords can work either as

1. Definitions, for instance "define function $f(x)$ and read its data from file `f.dat`"), or as
2. Instructions, such as "write the stress at point D into the standard output".

A person can tell if a keyword is a definition or an instruction because the former are nouns (`FUNCTION`) and the latter verbs (`PRINT`). The equal sign `=` is a special keyword that is neither a verb nor a noun, and its meaning changes depending on what is on the left hand side of the assignment.

a. If there is a function, then it is a definition: define an algebraic function to be equal to the expression on the right-hand side, e.g.:

   ```
   f(x,y) = exp(-x^2)*cos(pi*y)
   ```

b. If there is a variable, vector or matrix, it is an instruction: evaluate the expression on the right-hand side and assign it to the variable or vector (or matrix) element indicated in the left-hand side. Strictly speaking, if the variable has not already been defined (and implicit declaration is allowed), then the variable is also defined as well, e.g:

   ```
   VAR a
   VECTOR b[3]
   a = sqrt(2)
   b[i] = a*i^2
   ```

   There is no need to explicitly define the scalar variable `a` with `VAR` since the first assignment also defines it implicitly (if this is allowed by the keyword `IMPLICIT`).

An input file can define its own variables as needed, such as `my_var` or `flag`. But there are some reserved names that are special in the sense that they either

1. can be set to modify the behavior of FeenoX, such as `max_dt` or `dae_tol`
2. can be read to get the internal status or results back from FeenoX, such as `nodes` or `keff`
3. can be either set or read, such as `dt` or `done`

The problem being solved can be static or transient, depending on whether the special variable `end_time` is zero (default) or not. If it is zero and `static_steps` is equal to one (default), the instructions in the input file are executed once and then FeenoX quits. For example

```
VAR x
PRINT %.7f func_min(cos(x)+1,x,0,6)
```

If `static_steps` is larger than one, the special variable `step_static` is increased and they are repeated the number of time indicated by `static_steps`:

```
static_steps = 10
f(n) = n^2 - n + 41
PRINT f(step_static^2-1)
```

If the special variable `end_time` is set to a non-zero value, after computing the static part a transient problem is solved. There are three kinds of transient problems:

1. Plain "standalone" transients
2. Differential-Algebraic equations (DAE) transients
3. Partial Differential equations (PDE) transients

In the first case, after all the instruction in the input file were executed, the special variable `t` is increased by the value of `dt` and then the instructions are executed all over again, until `t` reaches `end_time`:

```
end_time = 2*pi
dt = 1/10

y = lag(heaviside(t-1), 1)
z = random_gauss(0, sqrt(2)/10)

PRINT t sin(t) cos(t) y z HEADER
```

In the second case, the keyword `PHASE_SPACE` sets up DAE system. Then, one initial condition and one differential-algebraic equation has to be given for each element in the phase space. The instructions before the DAE block executed, then the DAE timestep is advanced and finally the instructions after DAE block are executed (there cannot be any instruction between the first and the last DAE):

```
PHASE_SPACE x
end_time = 1
x_0 = 1
x_dot = -x
PRINT t x exp(-t) HEADER
```

The timestep is chosen by the SUNDIALS library in order to keep an estimate of the residual error below `dae_tol` (default is 10^{-6}), although `min_dt` and `max_dt` can be used to control it. See the section of the [Differential-Algebraic Equations subsystem] for more information.

In the third cae, the type of PDE being solved is given by the keyword `PROBLEM`. Some types of PDEs do support transient problems (such as `thermal`) but some others do not (such as `modal`). See the detailed explanation of each problem type for details. Now the transient problem is handled by the TS framework of the PETSc library. In general transient PDEs involve a mesh, material properties, initial conditions, transient boundary conditions, etc. And they create a lot of data since results mean spatial and temporal distributions of one or more scalar fields:

```
# example of a 1D heat transient problem
# from https://www.mcs.anl.gov/petsc/petsc-current/src/ts/tutorials/ex3.c.html
# a non-dimensional slab 0 < x < 1 is kept at T(0) = T(1) = 0
# there is an initial non-trivial T(x)
# the steady-state is T(x) = 0
PROBLEM thermal 1d
READ_MESH slab60.msh

end_time = 1e-1

# initial condition
T_0(x) := sin(6*pi*x) + 3*sin(2*pi*x)
# analytical solution
T_a(x,t) := exp(-36*pi^2*t)*sin(6*pi*x) + 3*exp(-4*pi^2*t)*sin(2*pi*x)
```

```
# unitary non-dimensional properties
k = 1
rho = 1
cp = 1

# boundary conditions
BC left  T=0
BC right T=0

SOLVE_PROBLEM

PRINT %e t dt T(0.1) T_a(0.1,t) T(0.7) T_a(0.7,t)
WRITE_MESH temp-slab.msh T

IF done
 PRINT "\# open temp-anim-slab.geo in Gmsh to see the result!"
ENDIF
```

PETSc's TS also honors the `min_dt` and `max_dt` variables, but the time step is controlled by the allowed relative error with the special variable `ts_rtol`. Again, see the section of the [Partial Differential Equations subsystem] for more information.

6.1 Algebraic expressions

To be done.

- Everything is an expression.

6.2 Initial conditions

6.3 Expansions of command line arguments

7 FeenoX & the Unix Philospohy

In 1978, Doug McIlroy—the inventor of Unix pipes and one of the founders of the Unix tradition—stated:

1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new features.
2. 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.
3. Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.
4. Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.

He later summarized it this way:

> This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

FeenoX explicitly followed the above ideas from scratch, especially the for sentences in bullet ii. It is even, like Unix itself, a third-system effect where clumsy parts of previous attempts were thrown away and rebuilt from scratch. The following sections explain how each of the seventeen rules was taken into account when designing and implementing FeenoX.

7.1 Rule of Modularity

> Developers should build a program out of simple parts connected by well defined interfaces, so problems are local, and parts of the program can be replaced in future versions to support new features. This rule aims to save time on debugging code that is complex, long, and unreadable.

FeenoX is designed to be as lightweight as possible. On the one hand, it relies on third-party high-quality libraries to do the heavy mathematical weightlifting such as

- GNU Scientific Library (`https://www.gnu.org/software/gsl/`) for general mathematics,
- SUNDIALS IDA (`https://computing.llnl.gov/projects/sundials/ida`) for ODEs and DAEs,
- PETSc (`https://petsc.org/`) for linear, non-linear and transient PDEs, and
- SLEPc (`http://slepc.upv.es/`) for PDEs involving eigen problems

because these libraries were written by professional programmers using algorithms designed by professional mathematicians. Yet-to-be-discovered improved mathematical schemes and/or coding algorithms can be eventually used by FeenoX by just updating those dependencies, which for sure will keep their well-defined interfaces (because they are programmed by professional programmers).

Moreover, the extensibility feature (@sec:unix-extensibility) of having each PDE in separate directories which can be added or removed at compile time without changing any line

of the source code goes into this direction as well. Relying of C function pointers allows (in principle) to replace these "virtual" methods with other ones using the same interface.

> Note that our (human) languages in general and words in particular shape and limit the way we think. Fortran's concept of "modules" is *not* the same as Unix's concept of "modularity." I wish two different words had been used.

7.2 Rule of Clarity

> Developers should write programs as if the most important communication is to the developer who will read and maintain the program, rather than the computer. This rule aims to make code as readable and comprehensible as possible for whoever works on the code in the future.

Of course there might be a confirmation bias in this section because every programmer thinks their code is clear (and everybody else's is not). But the first design decision to fulfill this rule is the programming language: there is little change to fulfill it with Fortran. One might argue that C++ can be clearer than C in some points, but for the vast majority of the source code they are equally clear. Besides, C is far simpler than C++ (see rule of simplicity).

The second decision is not about the FeenoX source code but about FeenoX inputs: clear human-readable input files without any extra unneeded computer-level nonsense. The two illustrative cases are the NAFEMS LE10 (`https://www.seamplex.com/feenox/examples/mechanical.html#nafems-le10-thick-plate-pressure-benchmark`) & LE11 (`https://www.seamplex.com/feenox/examples/mechanical.html#nafems-le11-solid-cylindertapersphere-temperature-benchmark`) benchmarks, where there is a clear one-to-one correspondence between the "engineering" formulation and the input file FeenoX understands.

7.3 Rule of Composition

> Developers should write programs that can communicate easily with other programs. This rule aims to allow developers to break down projects into small, simple programs rather than overly complex monolithic programs.

Previous designs of FeenoX' predecessors used to include instructions to perform parametric sweeps(and even optimization loops), non-trivial macro expansions using M4 and even execution of arbitrary shell commands. These non-trivial operations were removed from FeenoX to focus on the rule of composition, paying especially attention to easing the inclusion of calling the `feenox` binary from shell scripts, enforcing the composition with other Unix-like tools. Emphasis has been put on adding flexibility to programmatic generation of input files (see also rule of generation in @sec:unix-generation) and the handling and expansion of command-line arguments to increase the composition with other programs.

Moreover, the output is 100% controlled by the user at run-time so it can be tailored to suit any other programs' input needs as well. An illustrative example is creating professional-looking tables with results using AWK & LaTeX (`https://www.seamplex.com/feenox/doc/sds.html#sec:interoperability`).

7.4 Rule of Separation

> Developers should separate the mechanisms of the programs from the policies of the programs; one method is to divide a program into a front-end interface and a back-end engine with which that interface communicates. This rule aims to prevent bug introduction by allowing policies to be changed with minimum likelihood of destabilizing operational mechanisms.

FeenoX relies of the rule of separation (which also links to the next two rules of simplicity and parsimony) from the very beginning of its design phase. It was explicitly designed as a glue layer between a mesher like Gmsh and a post-processor like Gnuplot, Gmsh or Paraview. This way, not only flexibility and diversity (see #sec:unix-diversity) can be boosted, but also technological changes can be embraced with little or no effort. For example, CAEplex (`https://www.caeplex.com`) provides a web-based platform for performing thermo-mechanical analysis on the cloud running from the browser. Had FeenoX been designed as a traditional desktop-GUI program, this would have been impossible. If in the future CAD/CAE interfaces migrate into virtual and/or augmented reality with interactive 3D holographic input/output devices, the development effort needed to use FeenoX as the back end is negligible.

7.5 Rule of Simplicity

> Developers should design for simplicity by looking for ways to break up program systems into small, straightforward cooperating pieces. This rule aims to discourage developers' affection for writing "intricate and beautiful complexities" that are in reality bug prone programs.

The main source of simplicity comes from the design of the syntax of the input files, discussed in detail in the SDS (`https://www.seamplex.com/feenox/doc/sds.html#sec:input`):

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

7.6 Rule of Parsimony

> Developers should avoid writing big programs. This rule aims to prevent overinvestment of development time in failed or suboptimal approaches caused by the owners of the program's reluctance to throw away visibly large pieces of work. Smaller programs are not only easier to write, optimize, and maintain; they are easier to delete when deprecated.

We already said that FeenoX is a glue layer between a mesher and a post-processing tool. Even more, at another level, it acts as two glue layers between the mesher and PETSc, and PETSc and the post-processor.

On the other hand, we also already stated that FeenoX was written from scratch after throwing away clumsy code from two previous attempts. For instance, these previous versions used to implement parametric and optimization schemes. Instead, in FeenoX, these type of runs have to be driven from an outer script (Bash, Python, etc.)

7.7 Rule of Transparency

> Developers should design for visibility and discoverability by writing in a way that their thought process can lucidly be seen by future developers working on the project and using input and output formats that make it easy to identify valid input and correct output. This rule aims to reduce debugging time and extend the lifespan of programs.

As with the rule of clarity (@sec:unix-clarity), there is a risk of falling into the confirmation bias because every programmer thinks its code is transparent. Anyway, FeenoX is written in C99 which is way easier to debug than both Fortran and C++. Yet, very much like PETSc, FeenoX makes use of structures and function pointers to give the same functionality as C++'s virtual methods without needing to introduce other complexities that make the code base harder to maintain and to debug.

Regarding identification of valid inputs and correct outputs,

1. The build system includes a `make check` target that runs hundreds of regressions tests (`https://github.com/seamplex/feenox/tree/main/tests`).
2. The code supports verification using the Method of Manufactured Solutions (`https://github.com/seamplex/feenox/tree/main/tests/mms`)

7.8 Rule of Robustness

> Developers should design robust programs by designing for transparency and discoverability, because code that is easy to understand is easier to stress test for unexpected conditions that may not be foreseeable in complex programs. This rule aims to help developers build robust, reliable products.

Robustness is the child of transparency and simplicity.

7.9 Rule of Representation

> Developers should choose to make data more complicated rather than the procedural logic of the program when faced with the choice, because it is easier for humans to understand complex data compared with complex logic. This rule aims to make programs more readable for any developer working on the project, which allows the program to be maintained.

There is a trade off between clarity and efficiency. However, avoiding Fortran should already fulfill this rule. FeenoX uses C structures with function pointers, which make it far simple to understand than similar Fortran-based FEM tools. Just compare the source directories of FeenoX and CalculiX. Take for instance the file `stressc` (`https://github.com/seamplex/feenox/blob/main/src/pdes/mechanical/stress.c`) from `src/pdes/mechanical` (`https://github.com/seamplex/feenox/blob/main/src/pdes/mechanical/stress.c`) (which if deleted, will remove support for `mechanical` problems but it will not prevent the compilation of `feenox`) from the former and `calcstressf` (`https://github.com/calculix/ccx_prool/blob/master/CalculiX/ccx_2.21/src/calcstress.f`) (buried inside 2,400 files in `src` (`https://github.com/calculix/ccx_prool/tree/master/CalculiX/ccx_2.21/src`)) from the latter. There might be more illustrative examples showing how FeenoX' design is more representative than of CalculiX, but it is way

too hard to understand the source code of the latter (even though the license is supposed to be GPL).

7.10 Rule of Least Surprise

> Developers should design programs that build on top of the potential users' expected knowledge; for example, '+' in a calculator program should always mean 'addition'. This rule aims to encourage developers to build intuitive products that are easy to use.

The rules of input syntax have been designed with this rule in mind. Just note a couple of them:

- The command-line arguments after the input file are available to be expanded verbatim in the input file as `$1`, `$2`, etc. (or `${1}`, `${2}`, etc. if they appear in the middle of strings). This syntax matches Bash' syntax for expanding command-line arguments, so any person reading an input file with this syntax already knows what it does. ´
- If one needs a problem where the conductivity depends on x as $k(x) = 1 + x$ then the input is

  ```
  k(x) = 1+x
  ```

- If a problem needs a temperature distribution given by an algebraic expression $T(x, y, z) = \sqrt{x^2 + y^2} + z$ then do

  ```
  T(x,y,z) = sqrt(x^2+y^2) + z
  ```

- This syntax for (basic) algebraic expressions matches the common syntax found in Gmsh, Maxima and many other scientific tools. More complex expressions (e.g. involving hyperbolic tangents) might differ slightly.

7.11 Rule of Silence

> Developers should design programs so that they do not print unnecessary output. This rule aims to allow other programs and developers to pick out the information they need from a program's output without having to parse verbosity.

TL;DR: no `PRINT` (or `WRITE_RESULTS`), no output.

7.12 Rule of Repair

> Developers should design programs that fail in a manner that is easy to localize and diagnose or in other words "fail noisily". This rule aims to prevent incorrect output from a program from becoming an input and corrupting the output of other code undetected.

Input errors are detected before the computation is started:

```
$ feenox thermal-error.fee
error: undefined thermal conductivity 'k'
$
```

Run-time errors (even inside the numerical libraries) are caught with custom handlers.

7.13 Rule of Economy

> Developers should value developer time over machine time, because machine cycles today are relatively inexpensive compared to prices in the 1970s. This rule aims to reduce development costs of projects.

As explained in the SDS (`https://www.seamplex.com/feenox/doc/sds.html#sec:output`), output is 100% user-defined so only the desired results are directly obtained instead of needing further digging into tons of undesired data. The approach of "compute and write everything you can in one single run" made sense in 1970 where CPU time was more expensive than human time, but not anymore. Once again, the iconic examples are the NAFEMS LE10 (`https://www.seamplex.com/feenox/examples/mechanical.html#nafems-le10-thick-plate-pressure-benchmark`) & LE11 (`https://www.seamplex.com/feenox/examples/mechanical.html#nafems-le11-solid-cylindertapersphere-temperature-benchmark`) benchmarks, where just the required scalar stress at the required location is written into the standard output.

7.14 Rule of Generation

> Developers should avoid writing code by hand and instead write abstract high-level programs that generate code. This rule aims to reduce human errors and save time.

Some key points:

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

More saliently, the automatic detection of the available PDEs in `src/pdes` is an example of this rule. The `autogen.sh` would loop over each subdirectory and create a source file `src/pdes/parser.c` with a function `feenox_pde_parse_problem_type()` which then will be part of the actual FeenoX source base as the entry point for parsing the `PROBLEM` keyword.

7.15 Rule of Optimization

> Developers should prototype software before polishing it. This rule aims to prevent developers from spending too much time for marginal gains.

FeenoX is still "premature" for heavy optimization. Yet, it is (relatively) faster than other alternatives. It does use link-time optimization to allow for inlining of small routines. There is even a FeenoX benchmarking repository that uses Google's Benchmark library to prototype code optimization: `https://github.com/seamplex/feenox-benchmark`.

7.16 Rule of Diversity

> Developers should design their programs to be flexible and open. This rule aims to make programs flexible, allowing them to be used in ways other than those their developers intended.

FeenoX can read Gmsh files, but they need not necessarily be created by Gmsh. Other meshing formats (VTK with group names?) are planned to be implemented. Also, either Gmsh or Paraview can be used to post-process results. But also other formats are planned. See @sec:unix-extensibility. Diversity is embraced from the bottom up!

7.17 Rule of Extensibility

> Developers should design for the future by making their protocols extensible, allowing for easy plugins without modification to the program's architecture by other developers, noting the version of the program, and more. This rule aims to extend the lifespan and enhance the utility of the code the developer writes.

The main extensibility feature is that each PDE has a separate source directory. Any of them can be used as as template to add new PDEs, which are detected at compile time by the Autotools bootstrapping script.

A final note is that FeenoX is GPLv3+. First, this means that extensions and contributions are welcome. Each author retains the copyright on the contributed code (as long as it is free software). Second, the + is there for the future.