FeenoX

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

Contents

1 Why FeenoX?

> If by "Why FeenoX?" you mean "Why is FeenoX named that way?," read the FAQs.

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? Here is *__why FeenoX is different__.

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

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

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

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

Therefore, on the one hand, FeenoX is—in a certain sense—to desktop FEA programs (like Code_Aster with Salome-Meca or CalculiX with PrePoMax) and libraries (like MoFEM or Sparselizard) what Markdown is to Word and (La)TeX, respectively and *deliberately*.

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

2 How is FeenoX different?

FeenoX meets a fictitious-yet-plausible Software Requirement Specifications that no other single tool (that I am aware of) meets completely. The FeenoX Software Design Specifications address each requirement of the SRS. Two of the most important design-basis features are that FeenoX is…

1. a cloud-native computational tool (not just cloud *friendly,* but cloud **first**).
2. both free (as in freedom) and open source. See Licensing.

But the most important idea is that 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. FeenoX will provide means to

- parse the input file, handle command-line arguments, read mesh files, assign variables, evaluate conditionals, write results, etc. `feenox PROBLEM laplace 3D 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 at gauss points or at arbitrary locations for computing elementary contributions to
  - stiffness matrix
  - mass matrix
  - right-hand side vector
- 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

This general framework constitutes the bulk of FeenoX source code. The particular functions that implement each problem type are located in subdirectories `src/pdes`, namely

Engineers, researchers, scientists, developers and/or hobbyists just need to use 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.

They also ought to to read, understand and mind the GPLv3+ licensing terms.

3 What is FeenoX anyway?

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, or
- steady or quasi-static thermo-mechanical problems, or
- steady or transient heat conduction problems, or
- modal analysis problems, or
- neutron diffusion or transport problems, 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 (or any other pre-processing tool and converted to `.msh` format with Meshio for example). This assumption follows the *rule of composition* and prevents the actual input file to be polluted with mesh-dependent data (such as node coordinates and/or nodal loads) so as to keep it simple and make it Git-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
$
```

For example, let us consider the famous chaotic Lorenz's dynamical system. Here is one way of getting an image of the butterfly-shaped attractor using FeenoX to compute it and Gnuplot to draw it. Solve

$$\begin{cases} \dot{x} &= \sigma \cdot (y - x) \\ \dot{y} &= x \cdot (r - z) - y \\ \dot{z} &= xy - bz \end{cases}$$

for $0 < t < 40$ with initial conditions

$$\begin{cases} x(0) = -11 \\ y(0) = -16 \\ z(0) = 22.5 \end{cases}$$

and $\sigma = 10$, $r = 28$ and $b = 8/3$, which are the classical parameters that generate the butterfly as presented by Edward Lorenz back in his seminal 1963 paper Deterministic non-periodic flow.

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

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

sigma = 10            # the original parameters from the 1963 paper
r = 28
b = 8/3

x_0 = -11             # initial conditions
y_0 = -16
z_0 = 22.5

# the dynamical system's equations written as naturally as possible
x_dot = sigma*(y - x)
y_dot = x*(r - z) - y
z_dot = x*y - b*z

PRINT t x y z        # four-column plain-ASCII output
```

Indeed, when executing FeenoX with this input file, we get four ASCII columns (t, x, y and z) which we can then redirect to a file and plot it with a standard tool such as Gnuplot. Note the importance of relying on plain ASCII text formats both for input and output, as recommended by the UNIX philosophy and the *rule of composition*: other programs can easily create inputs for FeenoX and other programs can easily understand FeenoX's outputs. This is essentially how UNIX filters and pipes work.

Let us solve the linear elasticity benchmark problem NAFEMS LE10 "Thick plate pressure." Assuming a proper mesh has already been created in Gmsh, note how well the FeenoX input file matches the problem statement from fig. 2:

```
# 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
```

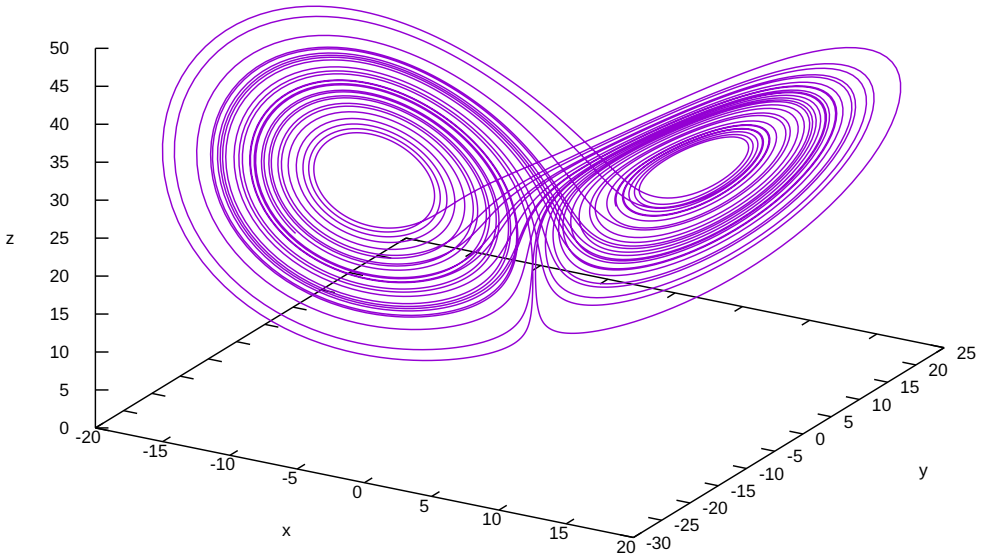

Figure 1: The Lorenz attractor solved with FeenoX and drawn with Gnuplot

```
# 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"
```

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.

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
```

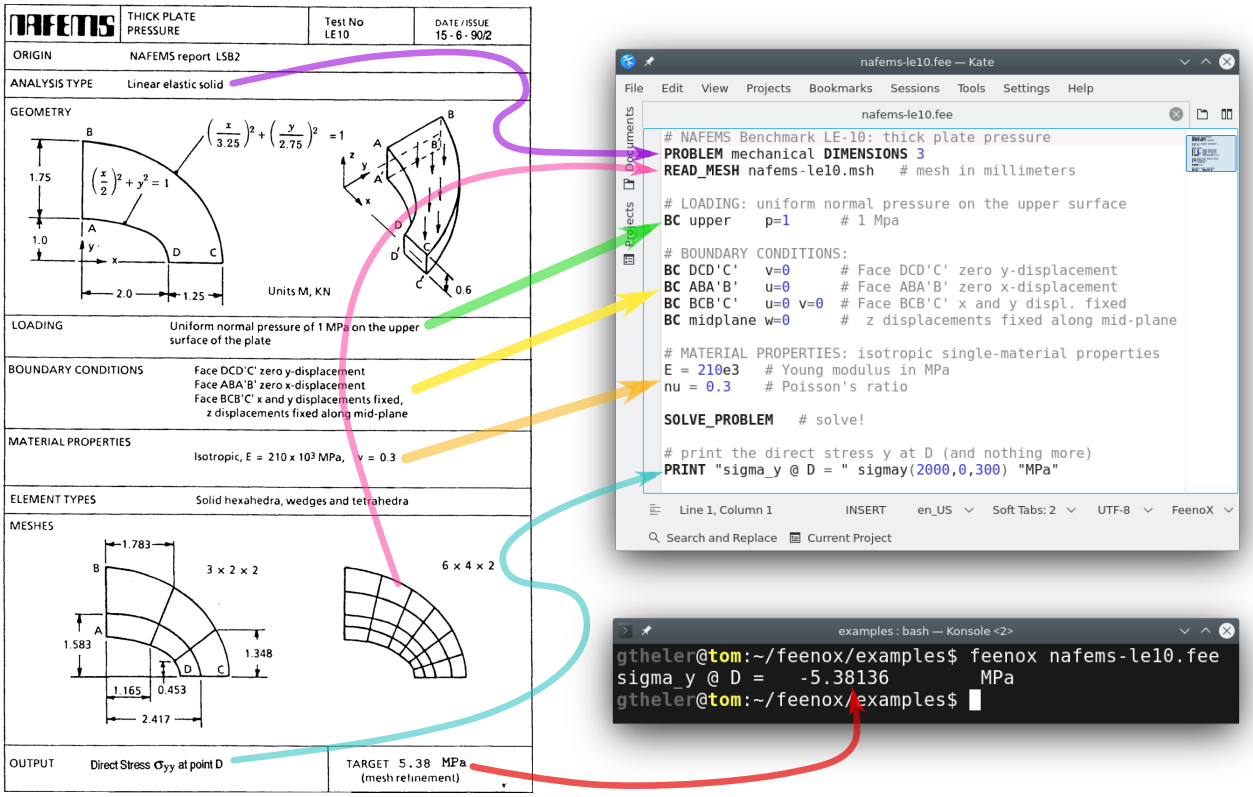

Figure 2: The NAFEMS LE10 problem statement and the corresponding FeenoX input

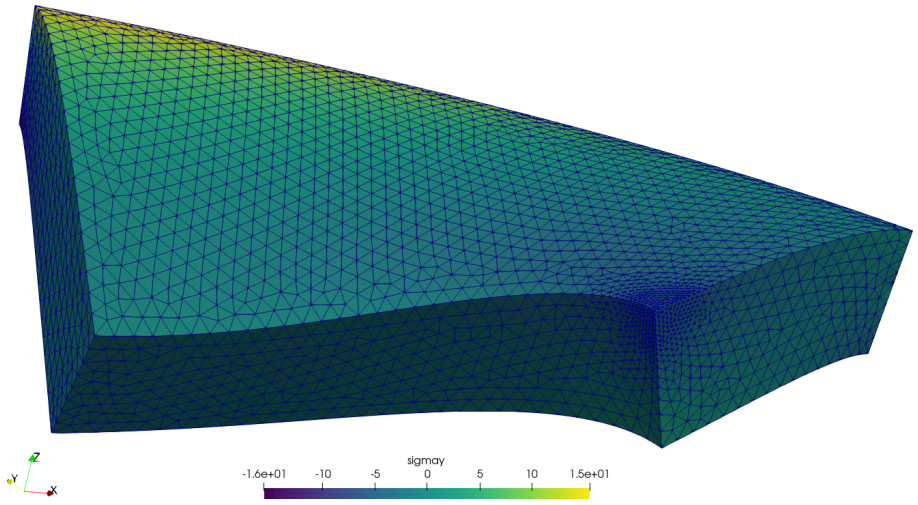

Figure 3: Normal stress σ_y refined around point D over 5,000x-warped displacements for LE10 created with Paraview

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

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 and Paraview respectively. See also CAEplex for a web-based interface.

- The input files should be syntactically sugared 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 and the Software Development Specification for details.

Since it is free (as in freedom) 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 for details about how to contribute.

4 Download

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

Debian/Ubuntu packages (unofficial)	https://www.seamplex.com/feenox/dist/deb
GNU/Linux static binaries	https://www.seamplex.com/feenox/dist/linux
Windows binaries	https://www.seamplex.com/feenox/dist/windows
Source tarballs	https://www.seamplex.com/feenox/dist/src
Github repository	https://github.com/seamplex/feenox/

- Be aware that FeenoX is a backend. It **does not have a GUI**. Read the documentation, especially the description and the FAQs. Ask for help on the GitHub discussions page.
- Debian/Ubuntu packages are unofficial, i.e. they are not available in `apt` repositories. They contain dynamically-linked binaries and their dependencies are hard-coded for each Debian/Ubuntu release. Make sure you get the right `.deb` for your release (i.e. `bookworm`/`bullseye` for Debian, `kinetic`/`focal` for Ubuntu).
- Generic GNU/Linux binaries are provided as statically-linked executables for convenience. They do not support MUMPS nor MPI and have only basic optimization flags. Please compile from source for high-end applications. See detailed compilatation instructions.
- Try to avoid Windows as much as you can. The binaries are provided as transitional packages for people that for some reason still use such an outdated, anachronous, awful and invasive operating system. They are compiled with Cygwin and have no support whatsoever. Really, really, **get rid of Windows ASAP**.

  > "It is really worth any amount of time and effort to get away from Windows if you are doing computational science."
  >
  > https://lists.mcs.anl.gov/pipermail/petsc-users/2015-July/026388.html

4.1 Git repository

To compile the Git repository, proceed as follows. This procedure does need `git` and `autoconf` but new versions can be pulled and recompiled easily. If something goes wrong and you get an error, do not hesitate to ask in FeenoX's discussion page.

1. Install mandatory dependencies

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

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

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

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

3. Clone Github repository

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

4. Boostrap, configure, compile & make

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

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

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

If you do not have Internet access, get the tarball manually, copy it to the same directory as `configure` and run again. See the detailed compilation instructions for an explanation.

5. Run test suite (optional)

```
make check
```

6. Install the binary system wide (optional)

```
sudo make install
```

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

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

See the detailed compilation instructions for further details.

5 Licensing

FeenoX is distributed under the terms of the GNU General Public License version 3 or (at your option) any later version. The following text was borrowed from the Gmsh documentation. Replacing “Gmsh” with “FeenoX” gives:

> FeenoX is "free software"; this means that everyone is free to use it and to redistribute it on a free basis. FeenoX is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of FeenoX that they might get from you.
>
> Specifically, we want to make sure that you have the right to give away copies of FeenoX, that you receive source code or else can get it if you want it, that you can change FeenoX or use pieces of FeenoX in new free programs, and that you know you can do these things.
>
> To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of FeenoX, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.
>
> Also, for our own protection, we must make certain that everyone finds out that there is no warranty for FeenoX. If FeenoX is modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.
>
> The precise conditions of the license for FeenoX are found in the General Public License that accompanies the source code. Further information about this license is available from the GNU Project webpage http://www.gnu.org/copyleft/gpl-faq.html.

FeenoX is licensed under the terms of the GNU General Public License version 3 or, at the user convenience, any later version. This means that users get the four essential freedoms:[3]

0. The freedom to *run* the program as they wish, for *any* purpose.
1. The freedom to *study* how the program works, and *change* it so it does their computing as they wish.
2. The freedom to *redistribute* copies so they can help others.
3. The freedom to *distribute* copies of their *modified* versions to others.

So a free program has to be open source, but it also has to explicitly provide the four freedoms above both through the written license and through the mechanisms available to get, modify, compile, run and document these modifications. That is why licensing FeenoX as GPLv3+ also implies that the source code and all the scripts and makefiles needed to compile and run it are available for anyone that requires it. Anyone wanting to modify the program either to fix bugs, improve it or add new features is free to do so. And if they do not know how to program, the have the freedom to hire a programmer to do it without needing to ask permission to the original authors.

Nevertheless, since these original authors are the copyright holders, they still can use it to either enforce

[3] There are some examples of pieces of computational software which are described as "open source" in which even the first of the four freedoms is denied. The most iconic case is that of Android, whose sources are readily available online but there is no straightforward way of updating one's mobile phone firmware with a customized version, not to mention vendor and hardware lock ins and the possibility of bricking devices if something unexpected happens. In the nuclear industry, it is the case of a Monte Carlo particle-transport program that requests users to sign an agreement about the objective of its usage before allowing its execution. The software itself might be open source because the source code is provided after signing the agreement, but it is not free (as in freedom) at all.

or prevent further actions from the users that receive FeenoX under the GPLv3+. In particular, the license allows re-distribution of modified versions only if they are clearly marked as different from the original and only under the same terms of the GPLv3+. There are also some other subtle technicalities that need not be discussed here such as what constitutes a modified version (which cannot be redistributed under a different license) and what is an aggregate (in which each part be distributed under different licenses) and about usage over a network and the possibility of using AGPL instead of GPL to further enforce freedom (TL;DR: it does not matter for FeenoX), but which are already taken into account in FeenoX licensing scheme.

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

6 Further information

Home page: https://www.seamplex.com/feenox
Repository: https://github.com/seamplex/feenox
Bug reporting: https://github.com/seamplex/feenox/issues
Discussions: https://github.com/seamplex/feenox/discussions
Follow us: YouTube LinkedIn Github
