A free and open source computational tool for solving differential equations in the cloud

Jeremy Theler

Mid-term evaluation, PhD in Nuclear Engineering
Instituto Balseiro, San Carlos de Bariloche, Argentina

be93413—2021-08-26

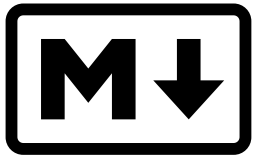
FeenoX [1]

a free no-fee no-X uniX-like
finite-element(ish)
computational engineering tool

Sources and full examples at https://github.com/gtheler/2021-presentation

[1] FeenoX needs a logo.

How do we write papers/reports/documents?

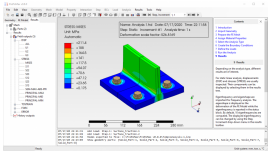


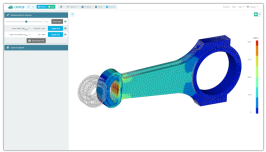
TEX

Feature	Word	Docs	Markdown*	TEX
Aesthetics	×	×	✓	✓
Convertibility	∼	∼	✓	∼
Traceability	×	∼	✓	✓
Mobile-friendliness	×	✓	✓	×
Collaborative	×	✓	✓	∼
Licensing/openness	×	×	✓	✓
Non-nerd friendliness	✓	✓	∼	×

* Markdown + Pandoc + Git + Github / Gitlab / Gitea

How do we do scientific/engineering computations?





FeenoX

a free no-fee no-X uniX-like finite-element(ish) computational engineering tool

deal.II SPACE-TIME MOFEM Firedrake

Feature	Desktop GUIs	Web frontends	FeenoX*	Libraries
Flexibility	∼	×	✓	✓
Scalability	×	∼	✓	✓
Traceability	×	∼	✓	✓
Cloud-friendliness	×	✓	✓	✓
Collaborative	×	✓	∼	×
Licensing/openness	✓/∼/×	×	✓	✓
Non-nerd friendliness	✓	✓	∼	×

* FeenoX + Gmsh + Paraview + Git + Github / Gitlab / Gitea

Software Requirement Specifications

After a successful project with a foreign company I decided to structure the PhD based on a fictitious & imaginary “Request for Quotation” for a computational tool:

1. Introduction
   - 1.1. Objective
   - 1.2. Scope
2. Architecture
   - 2.1. Deployment
   - 2.2. Execution
   - 2.3. Efficiency
   - 2.4. Scalability
   - 2.5. Flexibility
   - 2.6. Extensibility
   - 2.7. Interoperability
3. Interfaces
   - 3.1. Problem input
   - 3.2. Results output
4. Quality assurance
   - 4.1. Reproducibility and traceability
   - 4.2. Automated testing
   - 4.3. Bug reporting and tracking
   - 4.4. Verification
   - 4.5. Validation
   - 4.6. Documentation

FeenoX Software Design Specifications

- A fictitious & imaginary tender applying to the SRS addressing each section.

1. Introduction

- Application to industrial problems
  - Open source (to allow third-party V&V)
- First version should handle some problems
- Extensible to other problems & formulations
  - Free (as in freedom to hire somebody to modify/extend it)

1.1. Objective

- Solve DAEs and/or PDEs
  - Heat conduction
  - Elasticity
  - Electromagnetism
  - Fluid mechanics
  - ...
- State-of-the-art cloud friendly

FeenoX

- Free as “software libre”
  - GPLv3+
  - Only FOSS dependencies
  - Main target is `linux-x86_64`
  - Development environment is Debian
- Initial version supports
  - Dynamical systems (DAE)
  - Laplace/Poisson/Helmholtz (FEM)
  - Heat (FEM)
  - Elasticity (FEM)
  - Modal (FEM)
  - Neutron transport and diffusion (FEM/FVM)
- Templates for more formulations
  - Electromagnetism
  - Chemical diffusion/reaction
  - Fluid mechanics?

1.2. Scope

- The problem should be defined programatically
  - One or more input files (JSON, YAML, ad-hoc format), and/or
  - An API for high-level language (Python, Julia, etc.)
- There is no need to *include* a GUI
  - The tool should *allow* a GUI to be used
    - Desktop
    - Web
    - Mobile
- The mesh can be an input
  - As long as its creation meets the SRS
- Include documentation about how a...
  - Pre-processor should create inputs
  - Post-processor should read outputs

FeenoX

- No GUI, console binary executable
- "Transfer-function"-like between I/O
  - No need to recompile the binary

- English-like syntactic-sugared input files
  - Nouns are definitions
  - Verbs are instructions
- Python & Julia API: TO-DO
  - But already taken into account in the design & implementation
- Separate mesher
  - Gmsh (GPLv2, meets SRS)
  - Anything that writes `.msh`
- Possibility to use GUI
  - CAEplex https://www.caeplex.com

Transfer-function & English-like input: Lorenz' system

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

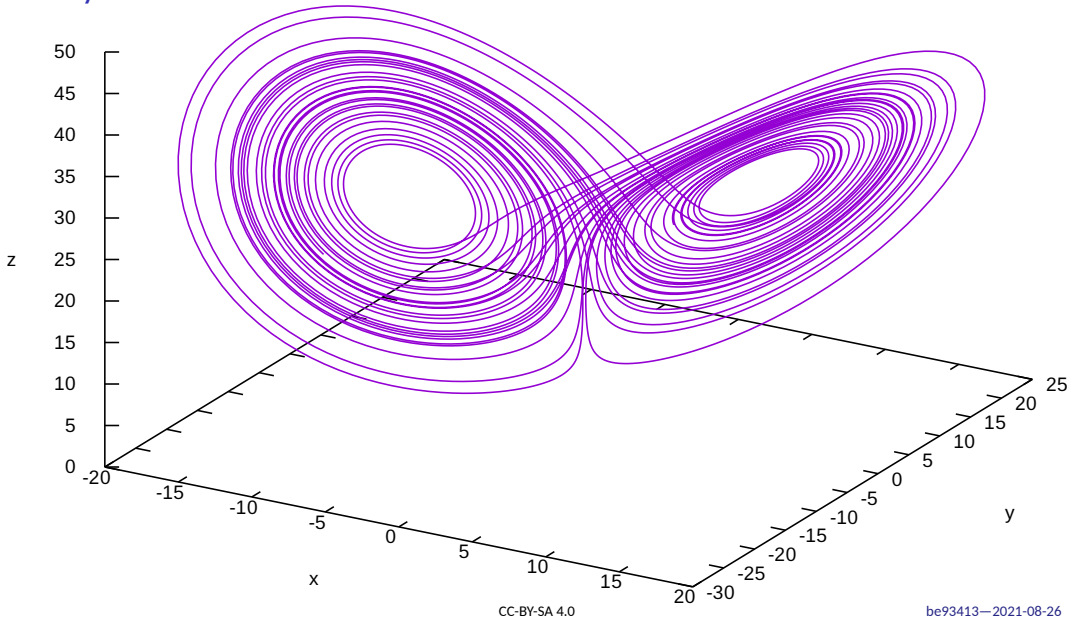
$$\begin{cases} x(0) = -11 \\ y(0) = -16 \\ z(0) = 22.5 \end{cases}$$

and $\sigma = 10, r = 28$ and $b = 8/3$.

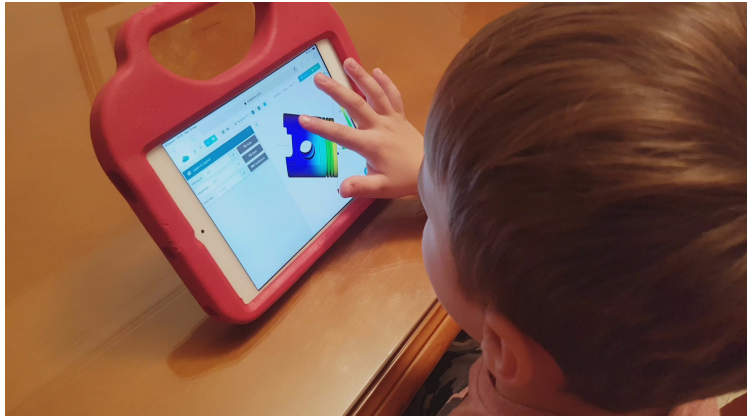
```
PHASE_SPACE x y z
end_time = 40          # dimensionless time

sigma = 10             # parameters
r = 28
b = 8/3

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

# Lorenz's equations as written in 1963
x_dot = sigma*(y - x)
y_dot = x*(r - z) - y
z_dot = x*y - b*z

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

```
$ feenox lorenz.fee
0.000000e+00    -1.100000e+01   -1.600000e+01   2.250000e+01
2.384186e-07    -1.100001e+01   -1.600001e+01   2.250003e+01
4.768372e-07    -1.100002e+01   -1.600002e+01   2.250006e+01
[...]
3.997567e+01    4.442995e+00    3.764391e+00    2.347301e+01
3.998290e+01    4.399950e+00    3.886609e+00    2.314602e+01
3.999012e+01    4.368713e+00    4.016860e+00    2.282821e+01
$
```

Lorenz' system



Web interface: CAEplex, finite elements in the cloud



https://www.seamplex.com/feenox/videos/caeplex-ipad.mp4

https://www.caeplex.com

2. Architecture

- Should run on mainstream cloud servers
  - GNU/Linux
  - Multi-core Intel-compatible CPUs
  - Several levels of memory cache
  - A few Gb of RAM
  - Several Gb of SSD
  - Either
    - Bare metal
    - Virtualized
    - Containerized
- Standard compilers, libraries and dependencies
  - Available in common GNU/Linux repositories
  - Preferable 100% open source
  - Adhere to well-established standards

FeenoX

- Third-system effect (after v1 & v2)
- UNIX philosophy: "do one thing well"
  - Rule of separation: no GUI
  - Rule of composition: Gnuplot, Gmsh, ...
  - ...more rules to come!
- Third-party math libraries
  - GNU GSL, PETSc, SLEPc, SUNDIALS
  - Rule of modularity
- Dependencies available in APT

```
apt-get install git gcc make automake autoconf
apt-get install libgsl-dev
apt-get install lib-sundials-dev petsc-dev slepc-dev
```

- Sources on github.com/seamplex/feenox

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

- Autotools & friends for compilation

```
./autogen.sh && ./configure && make
```

2. Architecture

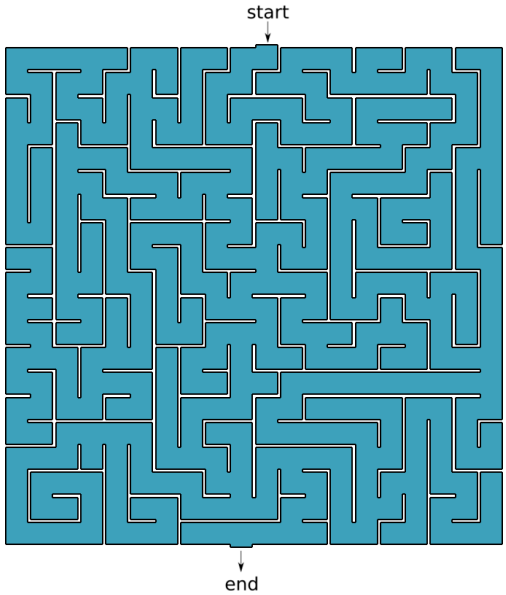
- Small coarse problems should be run in single hosts to check inputs
  - Local desktop/laptops (not needed but suggested)
  - Windows and MacOS (not needed but suggested)
  - Small cloud instances
- Large actual problems should be split into several hosts
  - HPC clusters
  - Scalable cloud instances
- Mobile devices (not needed but suggested)
  - As control/monitoring devices

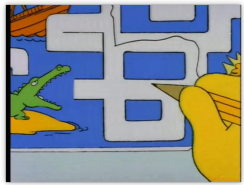
FeenoX

- Tested on
  - Raspberry Pi
  - Laptop (GNU/Linux & Windows 10)
  - Macbook
  - Desktop PC
  - Bare-metal servers
  - Vagrant/Virtualbox
  - Docker/Kubernetes
  - AWS/DigitalOcean/Contabo
- Parallelization: TO-DO
  - Gmsh partitioning with METIS
  - PETSc/SLEPc with MPI
- Web: https://www.caeplex.com (v2)

- Mobile: TO-DO

How to solve a maze without AI 1/3



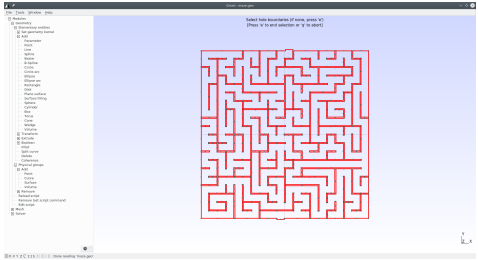


1. Go to http://www.mazegenerator.net/
2. Create a maze
3. Download it in PNG
4. Perform some conversions
   - PNG → PNM → SVG → DXF → GEO

```
$ wget http://www.mazegenerator.net/static/ ↩
      orthogonal_maze_with_20_by_20_cells.png
$ convert orthogonal_maze_with_20_by_20_cells.png \
  -negate maze.png
$ potrace maze.pnm --alphamax 0  --opttolerance 0 \
  -b svg -o maze.svg
$ ./svg2dxf maze.svg maze.dxf
$ ./dxf2geo maze.dxf 0.1
```

How to solve a maze without AI 2/3

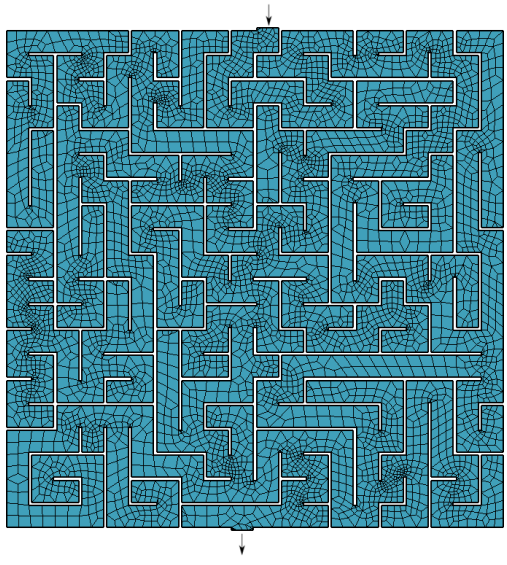
5 Open it with Gmsh



- Add a surface
- Set physical curves for “start” and “end”

6 Mesh it

```
gmsh -2 maze.geo
```



How to solve a maze without AI 3/3

7. Solve $\nabla^2\phi = 0$ with BCs

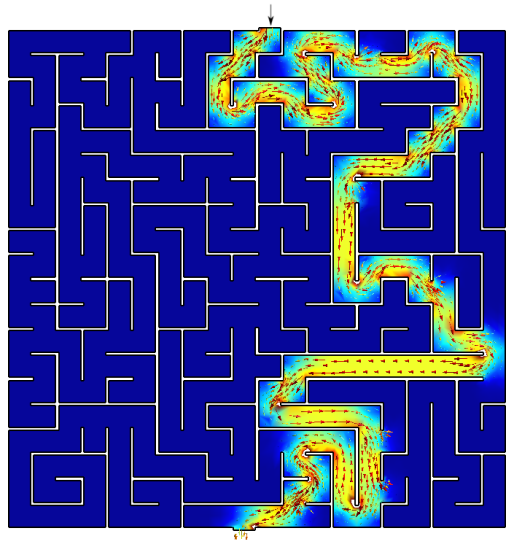
$$\begin{cases} \phi = 0 & \text{at “start”} \\ \phi = 1 & \text{at “end”} \\ \nabla\phi \cdot \hat{\mathbf{n}} = 0 & \text{everywhere else} \end{cases}$$

```
PROBLEM laplace 2D  # pretty self-descriptive, isn't it?
READ_MESH maze.msh

# boundary conditions (default is homogeneous Neumann)
BC start  phi=0
BC end    phi=1

SOLVE_PROBLEM

# write the norm of gradient as a scalar field
# and the gradient as a 2d vector into a .msh file
WRITE_MESH maze-solved.msh \
    sqrt(dphidx(x,y)^2+dphidy(x,y)^2) \
    VECTOR dphidx dphidy 0
```

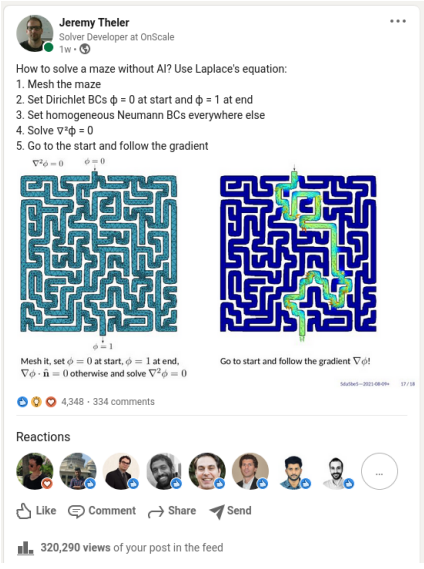
```
$ feenox maze.fee
$
```

(Rule of silence)

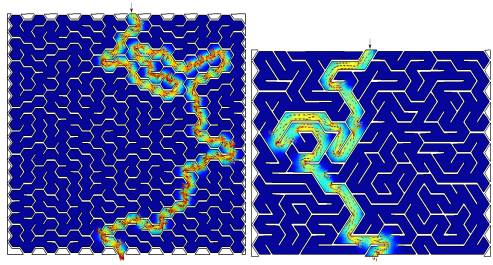
8. Go to start and follow the gradient $\nabla\phi$!

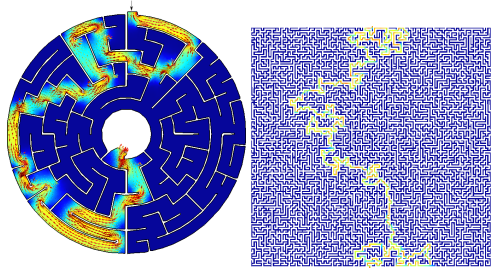


The life of an influencer…



http://www.mazegenerator.net/Examples.aspx





2.1. Deployment

- Automatically compile from source
  - Particular optimization flags
- Availability of pre-compiled binaries
  - Common architectures and options
- Both of them have to be available online

2.2. Execution

- Remote execution, either
  - By a direct user action
  - From a higher-level workflow
- Outer loops have to be supported
  - scripted
  - parametric
  - optimization
- Ways to read data from the outer loop
- Ways to write scalar figures of merit

FeenoX

- Compile optimized dependencies

```
$ cd $PETSC_DIR
$ export PETSC_ARCH=linux-fast
$ ./configure --with-debug=0 COPTFLAGS="-Ofast"
$ make -j8
```

- Configure FeenoX with particular flags

```
$ git clone https://github.com/seamplex/feenox
$ cd feenox
$ ./autogen.sh
$ export PETSC_ARCH=linux-fast
$ ./configure MPICH_CC=clang CFLAGS=-Ofast
$ make -j8
# make install
```

- Or use pre-compiled binaries

```
wget http://gmsh.info/bin/Linux/gmsh-Linux64.tgz
wget https://seamplex.com/feenox/dist/linux/feenox- ↩
       linux-amd64.tar.gz
```

- Everything is Docker-friendly
- Execution examples follow →

Direct execution: three ways of getting the first 20 Fibonacci numbers

```
# the Fibonacci sequence using the closed-form formula as a function
phi = (1+sqrt(5))/2
f(n) = (phi^n - (1-phi)^n)/sqrt(5)
PRINT_FUNCTION f MIN 1 MAX 20 STEP 1
```

```
# the fibonacci sequence as a vector
VECTOR f SIZE 20

f[i]<1:2> = 1
f[i]<3:vecsize(f)> = f[i-2] + f[i-1]

PRINT_VECTOR i f
```

```
# the fibonacci sequence as an iterative problem

static_steps = 20
#static_iterations = 1476  # limit of doubles

IF step_static=1|step_static=2
 f_n = 1
 f_nminus1 = 1
 f_nminus2 = 1
ELSE
 f_n = f_nminus1 + f_nminus2
 f_nminus2 = f_nminus1
 f_nminus1 = f_n
ENDIF

PRINT step_static f_n
```

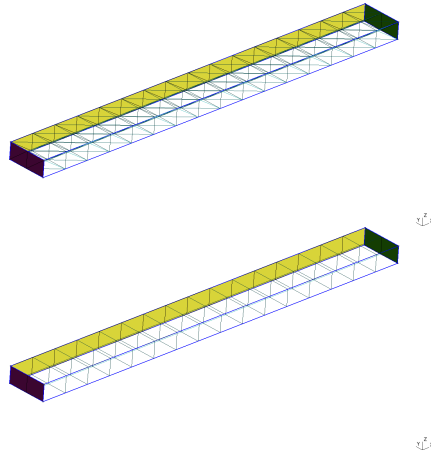
```
$ feenox fibo_formula.fee | tee one
1   1
2   1
3   2
4   3
5   5
6   8
7   13
8   21
9   34
10  55
11  89
12  144
13  233
14  377
15  610
16  987
17  1597
18  2584
19  4181
20  6765
$ feenox fibo_vector.fee > two
$ feenox fibo_iterative.fee > three
$ diff one two
$ diff two three
$
```

Parametric execution: shear locking in cantilevered beam

```
#!/bin/bash

rm -f *.dat
for element in tet4 tet10 hex8 hex20 hex27; do
 for c in $(seq 1 10); do

  # create mesh if not alreay cached
  mesh=cantilever-${element}-${c}
  if [ ! -e ${mesh}.msh ]; then
    scale=$(echo "PRINT 1/${c}" | feenox -)
    gmsh -3 -v 0 cantilever-${element}.geo -clscale ${scale} -o ${mesh}.msh
  fi

  # call FeenoX
  feenox cantilever.fee ${element} ${c} | tee -a cantilever-${element}.dat

 done
done
```

```
PROBLEM elastic 3D
READ_MESH cantilever-$1-$2.msh   # in meters

E = 2.1e11          # Young modulus in Pascals
nu = 0.3            # Poisson's ratio

BC left   fixed
BC right  tz=-1e5  # traction in Pascals, negative z

SOLVE_PROBLEM

# z-displacement (components are u,v,w) at the tip vs. number of nodes
PRINT nodes w(500,0,0) "\# $1 $2"
```

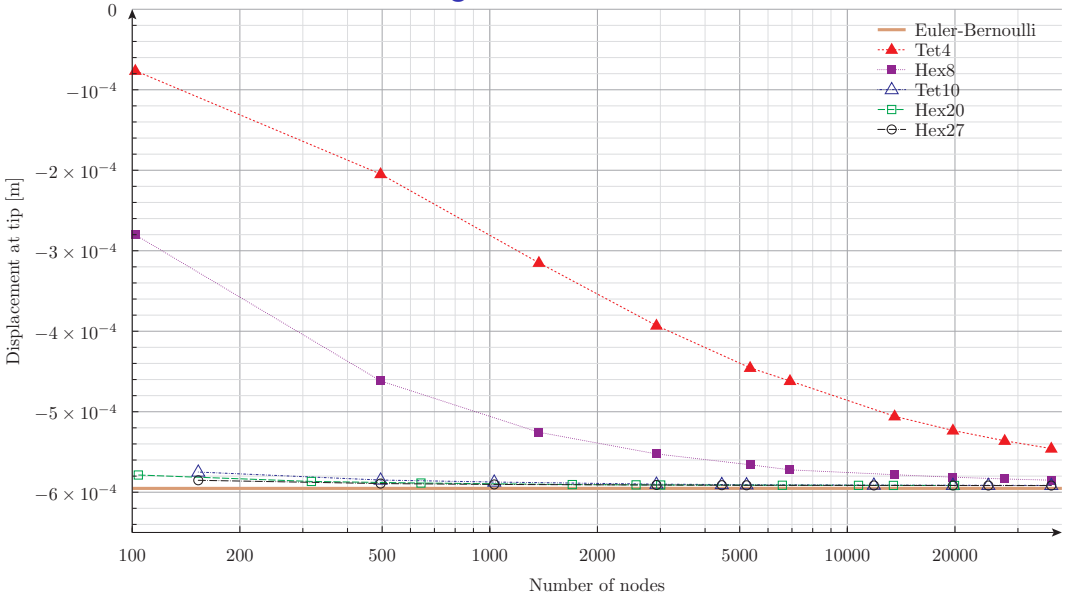
(Rule of generation)



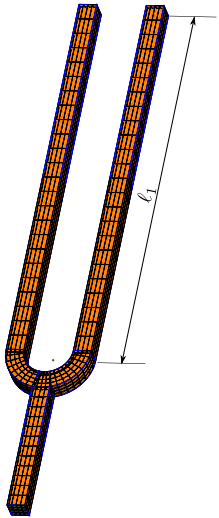
- Rule of simplicity
  - Only one material, no need to link volumes with materials

```
E = 2.1e11    # Young modulus in Pa
```

Parametric execution: shear locking in cantilevered beam



Optimization loop: finding the right length of a tuning fork



ℓ_1 to have 440 Hz?

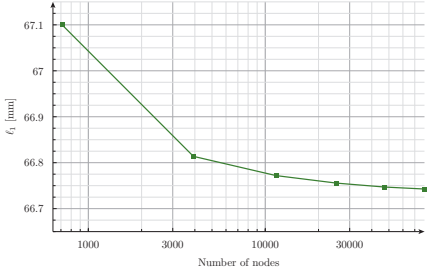
```python
import math
import gmsh
import subprocess  # to call FeenoX and read back

def create_mesh(r, w, l1, l2, n):
  gmsh.initialize()
  ...
  gmsh.finalize()
  return len(nodes)

def main():
  target = 440    # target frequency
  eps = 1e-2      # tolerance
  r = 4.2e-3      # geometric parameters
  w = 3e-3
  l1 = 30e-3
  l2 = 60e-3

  for n in range(1,7):   # mesh refinement level
    l1 = 60e-3              # restart l1 & error
    error = 60
    while abs(error) > eps:   # loop
      l1 = l1 - 1e-4*error
      # mesh with Gmsh Python API
      nodes = create_mesh(r, w, l1, l2, n)
      # call FeenoX and read scalar back
      # TODO: FeenoX Python API (like Gmsh)
      result = subprocess.run(['feenox', ↩
             'fork.fee'], stdout=subprocess.PIPE)
      freq = float(result.stdout.decode('utf-8'))
      error = target - freq

    print(nodes, l1, freq)
```

(Rule of parsimony)

```
PROBLEM modal 3D MODES 1  # only one mode needed
READ_MESH fork.msh  # in [m]
E = 2.07e11         # in [Pa]
nu = 0.33
rho = 7829          # in [kg/m^2]

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

# write back the fundamental frequency to stdout
PRINT f(1)
```

(Rule of simplicity)

```
$ python fork.py > fork.dat
$
```



2.3. Efficiency

- Similar to to other tools in terms of
  - CPU/GPU
  - RAM
  - Storage

2.4. Scalability

- Small problems to check correctness
- Large problems in parallel
  - Reasonable weak & strong scalability

2.5. Flexibility

- Engineering problems with
  - Multiple materials
  - Space-dependent properties
  - Space & time-dependent BCs
- Handle point-wise data
  - Properties
  - Time-dependent scalars

FeenoX

- First make it work, then optimize
  - Rule of optimization
- Premature optimization is the root of all evil
  - Optimization: TO-DO
  - Parallelization: TO-DO
  - Comparison: TO-DO
- Linear solvers
  - Direct solver MUMPS
    - Robust but not scalable
  - GAMG-preconditioned KSP
    - Near-nullspace improves convergence
- Non-linear & transient solvers
  - Scalable as PETSc
- Written in ANSI C99 (no C++ nor Fortran)
  - Autotools & friends, POSIX
  - Tested with `gcc`, `clang` and `icc`
  - Rust & Go, can't tell (yet)
  - Rule of transparency
- Flexibility follows $\rightarrow$

Flexibility I: one-dimensional thermal slab

Solve heat conduction on the slab $x \in [0:1]$ with boundary conditions

$$\begin{cases} T(0) = 0 & \text{(left)} \\ T(1) = 1 & \text{(right)} \end{cases}$$

and uniform conductivity. Compute $T\left(\frac{1}{2}\right)$.

- English self-evident ASCII input
  - Syntactic sugar
  - Simple problems, simple inputs
  - Robust (`heat` or `thermal`)
- Mesh separated from problem
  - Git-friendly `.geo` & `.fee`
- Output is 100% user-defined
  - No `PRINT` no output
  - Rule of silence
- There is no node at $x = 1/2 = 0.5$!

```
Point(1) = {0, 0, 0};          // geometry:
Point(2) = {1, 0, 0};          // two points
Line(1) = {1, 2};              // and a line connecting them!

Physical Point("left") = {1};  // groups for BCs and materials
Physical Point("right") = {2};
Physical Line("bulk") = {1};   // needed due to how Gmsh works

Mesh.MeshSizeMax = 1/3;        // mesh size, three line elements
Mesh.MeshSizeMin = Mesh.MeshSizeMax;
```

(Rule of composition)

```
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         # we are ready to solve the problem
PRINT T(1/2)          # ask for the temperature at x=1/2
```

(Rule of simplicity)

```
$ gmsh -1 slab.geo
[...]
Info    : 4 nodes 5 elements
Info    : Writing 'slab.msh'...
[...]
$ feenox thermal-1d-dirichlet-uniform-k.fee
0.5
$
```

(Rule of economy)

Flexibility II: one-dimensional thermal slabs

```
PROBLEM heat 1D
READ_MESH slab.msh

BC left  T=0
BC right T=1

INCLUDE $1.fee   # read k and solution from $1

SOLVE_PROBLEM

PRINT_FUNCTION T T_analytical MIN 0 MAX 1 NSTEPS 100
```

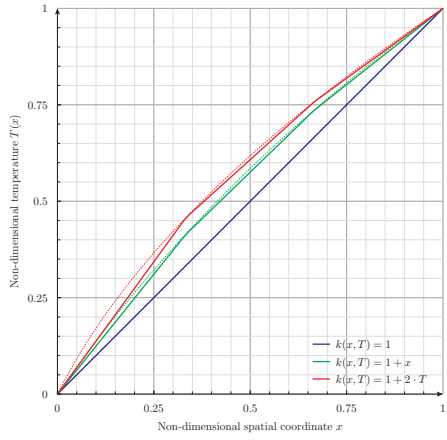
```
k = 1            # uniform.fee
T_analytical(x) = x
```

```
k(x) = 1+x       # space.fee
T_analytical(x) = log(1+x)/log(2)
```

```
a = 2            # temperature.fee
k(x) = 1+a*T(x)
T_analytical(x) = (1/a)*(sqrt(1+(2+a)*a*x)-1)
```

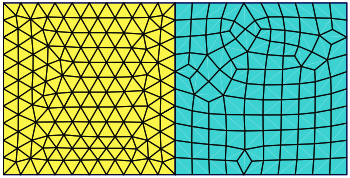
- Everything is an expression
- Similar problems need similar inputs
- Rule of least surprise: $k(x) = 1 + x$

```
for i in uniform space temperature; do
  feenox thermal-1d-dirichlet.fee ${i} > ${i}.dat
done
```



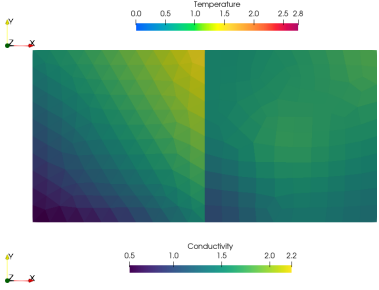
- FeenoX can tell that $k(T)$ is non-linear
  - It switchs from KSP to SNES

Flexibility III: two squares in thermal contact



```
PROBLEM thermal 2d
READ_MESH two-squares.msh

FUNCTION cond(x,y) INTERPOLATION shepard DATA {
    1   0    1.0
    1   1    1.5
    2   0    1.3
    2   1    1.8
    1.5 0.5  1.7 }

#        name    conductivity  power density
MATERIAL yellow k=0.5+T(x,y)  q=0
MATERIAL cyan   k=cond(x,y)   q=1-0.2*T(x,y)

BC left   T=y                 # temperature
BC bottom T=1-cos(pi/2*x)
BC right  q=2-y               # heat flux
BC top    q=1

SOLVE_PROBLEM

WRITE_MESH two-squares.vtk  T CELLS k
```

(Rule of clarity)

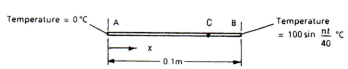


- Volumes ⇔ materials now needed
- FeenoX detects the problem is non-linear
- TO-DO: roughish output

Flexibility IV: thermal transient with time-dependent BCs

NAFEMS	ONE DIMENSIONAL TRANSIENT HEAT TRANSFER	Test No T3	DATE / ISSUE 15 - 6 - 90/2
ORIGIN	NAFEMS report BMTTA(S) & 'BENCHMARK' JAN, 1989 p23		
ANALYSIS TYPE	Transient heat conduction		

GEOMETRY



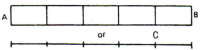
Uniform cross section

LOADING	Zero internal heat generation
BOUNDARY CONDITIONS	At time t = 0, All temperature = 0 °C At time t >0, At A, temperature = 0 °C At B, temperature = 100 sin °C No heat flux perpendicular to AB
MATERIAL PROPERTIES	Conductivity = 35.0 W/m° C Specific heat = 440.5J/kg° C Density = 7200kg/m³
ELEMENT TYPES	One or two-dimensional heat transfer elements

MESHES

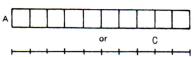
Coarse:

Uniform mesh of 5, 4-noded elements along length



Fine:

Uniform mesh of 10, 8-noded elements along length

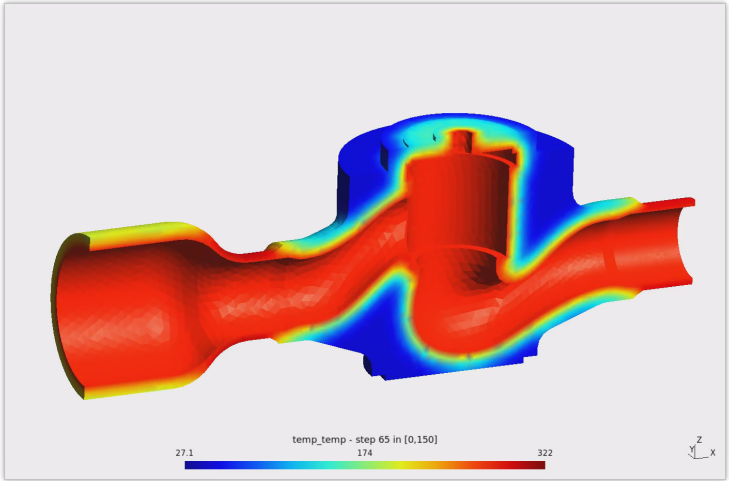


OUTPUT	Material temperature at point C, x = 0.08m, time t = 32 secs	TARGET 36.60°C

```
# NAFEMS-T3 benchmark: 1d transient heat conduction
PROBLEM heat DIMENSIONS 1
READ_MESH slab-0.1m.msh

T_0(x) = 0          # initial condition

BC left  T=0        # prescribed temperatures
BC right T=100*sin(pi*t/40)

k = 35.0            # conductivity [W/(m K)]
cp = 440.5          # heat capacity [J/(kg K)]
rho = 7200          # density [kg/m^3]

end_time = 32       # trasient up to 32 seconds

SOLVE_PROBLEM

PRINT %.3f t dt %.2f T(0.1) T(0.08)

IF done
 PRINT "\# result = " T(0.08) "ºC"
ENDIF
```

```
$ feenox nafems-t3.fee
0.000   0.062   0.00    0.00
0.002   0.002   0.01    0.00
[...]
30.871  0.565   65.71   36.04
31.435  0.565   62.31   36.33
32.000  1.050   58.78   36.56
# result =      36.5636 ºC
$
```

Flexibility V: 3D thermal transient with $k(\mathbf{x})$



https://www.seamplex.com/feenox/videos/temp-valve-smooth.mp4

Flexibility VI: point kinetics with point-wise reactivity

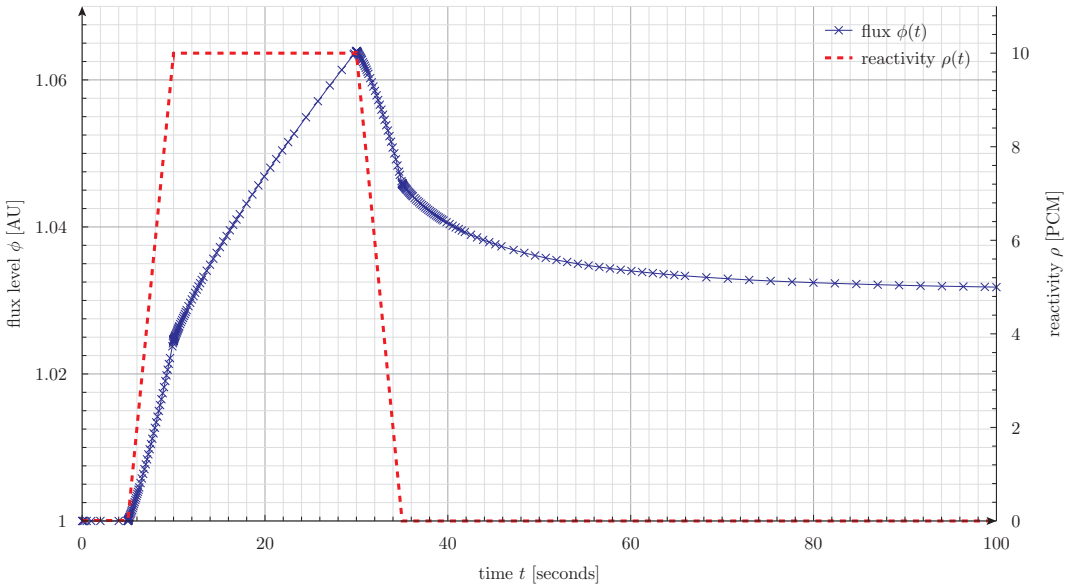
$$\begin{cases} \dot{\phi}(t) = \dfrac{\rho(t) - \mathrm{B}}{\Lambda} \cdot \phi(t) + \displaystyle\sum_{i=1}^{N} \lambda_i \cdot c_i \\ \dot{c}_i(t) = \dfrac{\beta_i}{\Lambda} \cdot \phi(t) - \lambda_i \cdot c_i \end{cases}$$

t [s]	$\rho(t)$ [pcm]
0	0
5	0
10	10
30	10
35	0
100	0

for $0 < t < 100$ starting from steady-steate conditions at full power.

```
INCLUDE parameters.fee    # kinetic parameters

# our phase space is flux, precursors and reactivity
PHASE_SPACE phi c rho

end_time = 100
# we need a tighter error to handle small reactivities
rel_error = 1e-7

# steady-state initial conditions
rho_0 = 0   # no reactivity
phi_0 = 1   # full power
c_0[i] = phi_0 * beta[i]/(Lambda*lambda[i])

FUNCTION react(t) DATA {   0    0   # in pcm
                           5    0
                           10  10
                           30  10
                           35   0
                           100  0   }

# reactor point kinetics equations
rho = 1e-5*react(t)   # convert pcm to absolute
phi_dot = (rho-Beta)/Lambda * phi + vecdot(lambda, c)
c_dot[i] = beta[i]/Lambda * phi - lambda[i]*c[i]

PRINT t phi rho
```

```
$ feenox reactivity-from-table.fee > flux.dat
$
```

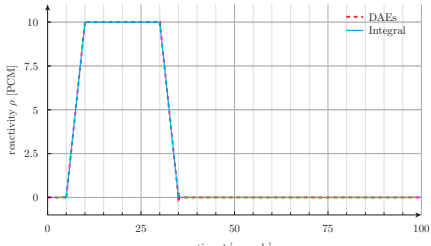

Point reactor equations with $\rho(t)$ given as scattered data
flux $\phi(t)$
reactivity $\rho(t)$
flux level ϕ [AU]
reactivity ρ [PCM]
time t [seconds]
1
1.02
1.04
1.06
0
2
4
6
8
10
0
20
40
60
80
100

Flexibility VI: inverse kinetics

```
INCLUDE parameters.fee
FUNCTION flux(t) FILE_PATH $1  # read flux from file

# define a flux function that allow for negative times
t_min = vec_flux_t[1]
t_max = vec_flux_t[vecsize(vec_flux)]
phi(t) := if(t<t_min, flux(t_min), flux(t))

VAR t'   # dummy integration variable
# compute the reactivity with the integral formula
rho(t) := Lambda * derivative(log(phi(t')),t',t) + Beta * ( 1  ↩
       - 1/phi(t) * integral(phi(t-t') *  ↩
       sum((lambda[i]*beta[i]/Beta)*exp(-lambda[i]*t'), i, 1,  ↩
       nprec), t', 0, 1e4) )

PRINT_FUNCTION rho MIN t_min MAX t_max STEP (t_max-t_min)/1000
```

```
INCLUDE parameters.fee
PHASE_SPACE phi c rho

end_time = 100
rel_error = 1e-7

rho_0 = 0
phi_0 = 1
c_0[i] = phi_0 * beta(i)/(Lambda*lambda(i))

FUNCTION flux(t) FILE $1
phi = flux(t) # force the variable phi to the data
phi_dot = (rho-Beta)/Lambda * phi + vecdot(lambda, c)
c_dot[i] = beta[i]/Lambda * phi - lambda[i]*c[i]

PRINT t phi rho
```

```
$ feenox inverse-dae.fee flux.dat > inverse-dae.dat
$ feenox inverse-integral.fee flux.dat > inverse-integral.dat
```



2.6. Extensibility

- Possibility to add more features
  - More PDEs
  - New material models (i.e. stress-strain)
  - Other element types
- Clear licensing scheme for extensions

2.7. Interoperability

- Ability to exchange data with other tools following this SRS
  - Pre and post processors
  - Optimization tools
  - Coupled multi-physics calculations

FeenoX

- Think for the future! Rule of extensibility
  - GPLv3+: the '+' is for the future
- Nice-to-haves: TO-DO
  - Lagrangian elements, DG, h-p AMR, ...
- Other problems & formulations: TO-DO
  - Each PDE has an independent directory
  - "Virtual methods" as function pointers
  - Use Laplace as a template (elliptic)
- Coupled calculations: TO-DO
  - Wide experience from CNA2 (v2)
  - Plain (RAM-disk) files
  - Shared memory & semaphores
  - MPI
- Interoperability
  - Gnuplot, matplotlib, etc.
  - Gmsh (+ Meshio), Paraview
  - CAEplex
  - PrePoMax, FreeCAD, ...: TO-DO

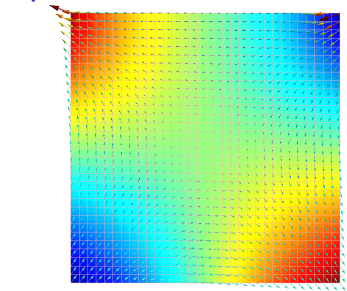
Laplace equation with both Gmsh & Paraview as post-processors

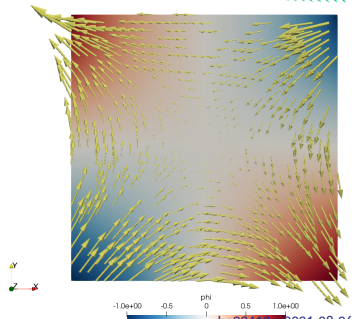
Solve $\nabla^2\phi = 0$ over $[-1:+1] \times [-1:+1]$ with

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

```
PROBLEM laplace 2d
READ_MESH square-centered.msh # [-1:+1]x[-1:+1]

# boundary conditions
BC left    phi=+y
BC right   phi=-y
BC bottom  dphidn=sin(pi/2*x)
BC top     dphidn=0

SOLVE_PROBLEM

# same output in .msh and in .vtk formats
WRITE_MESH laplace-square.msh phi VECTOR dphidx dphidy 0
WRITE_MESH laplace-square.vtk phi VECTOR dphidx dphidy 0
```

(Rule of diversity)





3. Interfaces

- Fully human-less execution
  - Input files (1 or more)
  - Output files (0 or more)
- Ability to remotely report status
  - Progress
  - Errors

3.1. Input

- Problem fully defined in input files
  - Ad-hoc syntax
  - API for high-level languages
  - Other files (data, meshes, scripts)
- Preferably ASCII (for DCVS)
  - Avoid mixing problem and mesh data
- GUI not mandatory but possible
  - Ok to have basic usage through GUI
  - Advanced features through API

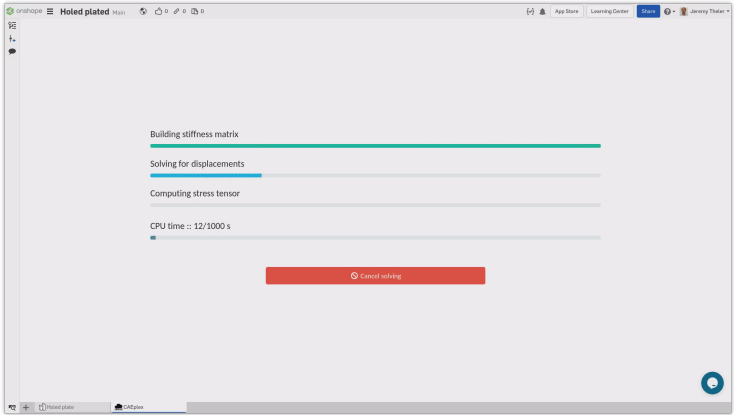
FeenoX

- Already deployed industrial human-less production workflow (based on v2)
- There are ASCII progress bars
  - Build matrix
  - Solve equations
  - Gradient recovery
- Heartbeat: TO-DO
- English self-evident ASCII input
  - Syntactically-sugared
    - Nouns are definitions
    - Verbs are instructions
  - Simple problems, simple inputs
  - Similar problems, similar inputs
  - Everything is an expression!
  - Rule of least surprise: $f(x) = \frac{1}{2} \cdot x^2$

```
f(x) = 1/2 * x^2
```

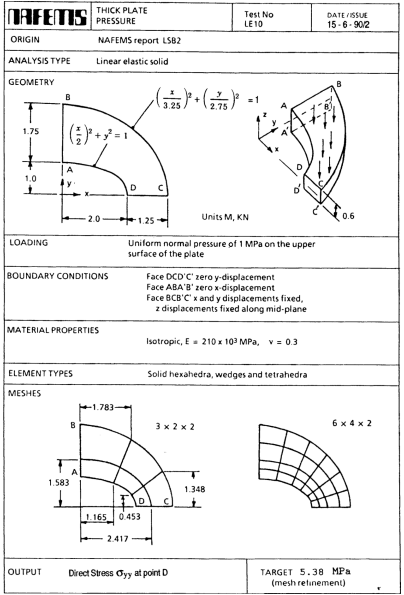
  - Expansion of command line arguments

CAEplex progress status on the cloud



https://www.seamplex.com/feenox/videos/caeplex-progress.mp4

NAFEMS LE10: English-like problem definition & user-defined output



NAFEMS	THICK PLATE PRESSURE	Test No LE10	DATE / ISSUE 15 · 6 · 90/2
ORIGIN	NAFEMS report LSB2		
ANALYSIS TYPE	Linear elastic solid		
GEOMETRY	$\left(\frac{x}{3.25}\right)^2+\left(\frac{y}{2.75}\right)^2=1$ $\left(\frac{x}{2}\right)^2+y^2=1$ 1.75, 1.0, 2.0, 1.25, 0.6 Units M, KN		
LOADING	Uniform normal pressure of 1 MPa on the upper surface of the plate		
BOUNDARY CONDITIONS	Face DCD'C' zero y-displacement Face ABA'B' zero x-displacement Face BCB'C' x and y displacements fixed, z displacements fixed along mid-plane		
MATERIAL PROPERTIES	Isotropic, E = 210 x 10^3 MPa, v = 0.3		
ELEMENT TYPES	Solid hexahedra, wedges and tetrahedra		
MESHES	1.783, 1.583, 1.348, 0.453, 1.165, 2.417 3 × 2 × 2 6 × 4 × 2		
OUTPUT	Direct Stress σ_{yy} at point D	TARGET 5.38 MPa (mesh refinement)	

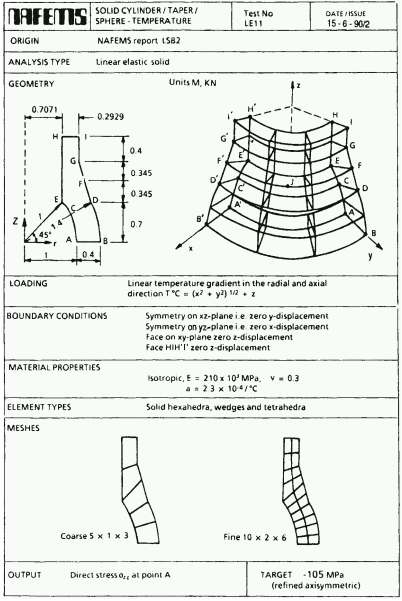
```
# NAFEMS Benchmark LE-10: thick plate pressure
PROBLEM mechanical DIMENSIONS 3
READ_MESH nafems-le10.msh   # mesh in millimeters

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

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

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

SOLVE_PROBLEM   # solve!

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

(Rule of clarity)

```
$ gmsh -3 nafems-le10.geo
[...]
Info    : Done meshing order 2 (Wall 0.433083s, CPU 0.414008s)
Info    : 205441 nodes 59892 elements
Info    : Writing 'nafems-le10.msh'...
$ feenox nafems-le10.fee
sigma_y @ D =  -5.38361 MPa
$
```

(Rule of economy)

NAFEMS LE11: everything is an expression (especially temperature)



NAFEMS	SOLID CYLINDER / TAPER / SPHERE - TEMPERATURE	Test No LE11	DATE / ISSUE 15 - 6 - 90/2
ORIGIN	NAFEMS report t.582		
ANALYSIS TYPE	Linear elastic solid		
GEOMETRY	Units M, KN		
LOADING	Linear temperature gradient in the radial and axial direction T °C = $(x^2 + y^2)^{1/2} + z$		
BOUNDARY CONDITIONS	Symmetry on xz-plane i.e. zero y-displacement Symmetry on yz-plane i.e. zero x-displacement Face on xy-plane zero z-displacement Face HIH'I' zero z-displacement		
MATERIAL PROPERTIES	Isotropic, E = 210 x 10^3 MPa, ν = 0.3 α = 2.3 x 10^{-4} / °C		
ELEMENT TYPES	Solid hexahedra, wedges and tetrahedra		
MESHES	Coarse 5 × 1 × 3 ; Fine 10 × 2 × 6		
OUTPUT	Direct stress σ_{zz} at point A	TARGET -105 MPa (refined axisymmetric)	

```
PROBLEM mechanical 3D
READ_MESH nafems-le11.msh

# linear temperature gradient in the radial and axial direction
T(x,y,z) := sqrt(x^2 + y^2) + z  # UTEMP for this????

BC xz      v=0       # displacement vector is [u,v,w]
BC yz      u=0       # u = displacement in x
BC xy      w=0       # v = displacement in y
BC HIH'I' w=0        # w = displacement in z

E = 210e3*1e6        # mesh is in meters, so E=210e3 MPa -> Pa
nu = 0.3             # dimensionless
alpha = 2.3e-4       # in 1/°C as in the problem
SOLVE_PROBLEM

# for post-processing in Paraview
WRITE_MESH nafems-le11.vtk VECTOR u v w   T sigmax sigmay sigmaz

PRINT "sigma_z(A) = " sigmaz(0,1,0)/1e6 "MPa" SEP " "
```

(Rule of least surprise)

```
$ gmsh -3 -clscale 0.5 nafems-le11.geo
[...]
Info    : 8326 nodes 1849 elements
Info    : Writing 'nafems-le11.msh'...
$ feenox nafems-le11.fee
sigma_z(A) =  -105.043 MPa
$
```

- 3D distributions in VTK

3.2. Output

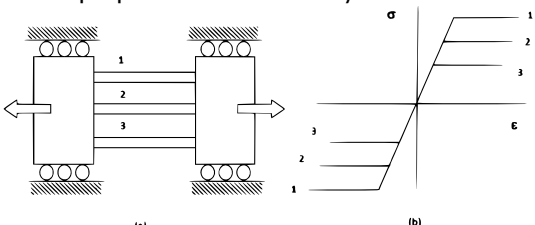
- Clean output expected
- Do not clutter the output with
  - ASCII art
  - Notices
  - Explanations
  - Page separators
- Output should interpreted by both
  - A human
  - A computer
- Open standards and well-documented formats should be preferred

FeenoX

- Rule of economy: output is completely defined by the user
- Rule of silence: no `PRINT` no output
- ASCII columns
  - `PRINT` & `PRINT_FUNCTION`
  - Gnuplot & compatible
  - Markdown/LaTeX tables
- Post-processing formats
  - `.msh`
  - `.vtk`
  - `.vtu`: TO-DO
  - `.hdf5`: TO-DO
  - `.frd`: TO-DO ?
- Dumping of vectors & matrices
  - ASCII
  - PETSc binary
  - Octave (sparse)

100% user-defined output: cycle loads

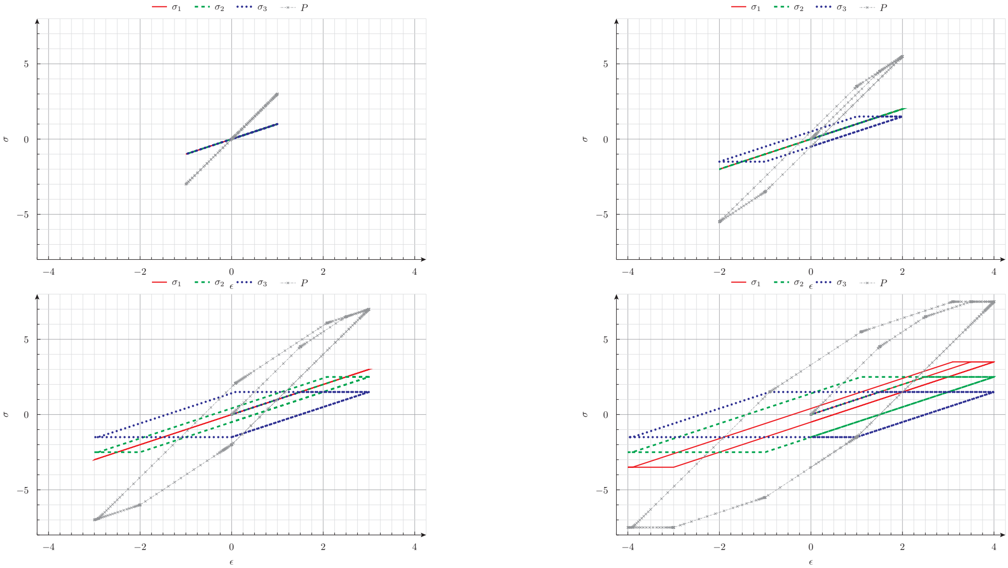
For the model below, draw the sequence of loading and unloading for different levels of strains. All bars have the same geometry and elastic properties but different yield stresses.




```
for i in 1 2 3 4; do
  feenox 3bars.fee ${i}
  pyxplot 3bars.ppl
  mv 3bars-sigma-vs-eps.pdf 3bars-sigma-vs-eps-${i}.pdf
done
```

```
DEFAULT_ARGUMENT_VALUE 1 5

E = 1          # non-dimensional Young modulus
yield1 = 3.5   # non-dimensional yield stresses
yield2 = 2.5
yield3 = 1.5

eps_max = $1   # max strain from command line
end_time = 2*eps_max

PHASE_SPACE eps sigma1 sigma2 sigma3 P

eps_0 = 0      # initial conditions
sigma1_0 = 0
sigma2_0 = 0
sigma3_0 = 0
P_0 = 0

# DAEs
eps = eps_max * sin(3*pi*t/end_time)
sigma1_dot = E * eps_dot * if((eps_dot < 0 | sigma1 < yield1)  ↩
       & (eps_dot > 0 | sigma1 > (-yield1)))
sigma2_dot = E * eps_dot * if((eps_dot < 0 | sigma2 < yield2)  ↩
       & (eps_dot > 0 | sigma2 > (-yield2)))
sigma3_dot = E * eps_dot * if((eps_dot < 0 | sigma3 < yield3)  ↩
       & (eps_dot > 0 | sigma3 > (-yield3)))
P = sigma1 + sigma2 + sigma3

PRINT FILE 3bars.dat t eps P sigma1 sigma2 sigma3
```

100% user-defined output: cycle loads



Markdown table: natural oscillation frequencies of a wire

Experimental Physics 101 (2004)

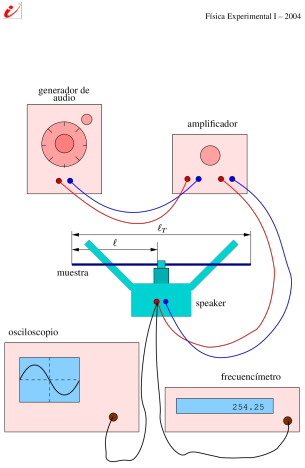


Figura 3: Disposición experimental. El osciloscopio en realidad es opcional, y sirve sólo para asegurarnos que la señal sinusoidal con la que alimentamos el speaker no sea deformada por alguna eventual saturación del amplificador.

```
# compare the frequencies
PRINT "  \$n\$ |   FEM  | Euler | Relative difference [%]"
PRINT ":----:+:------:+:-----:+:----------------------:"
PRINT_VECTOR i         f(2*i-1) f_euler   100*(f_euler(i)-f(2*i-1))/f_euler(i)
PRINT
PRINT ": $2 wire over $1 mesh, frequencies in Hz"
```

```
$ feenox wire.fee copper hex > copper-hex.md
$
```

Table 4: copper wire over hex mesh, frequencies in Hz

n	FEM	Euler	Relative difference [%]
1	45.8374	45.8448	0.0161707
2	287.126	287.302	0.0611787
3	803.369	804.454	0.134888
4	1572.59	1576.41	0.242324
5	2595.99	2605.92	0.381107

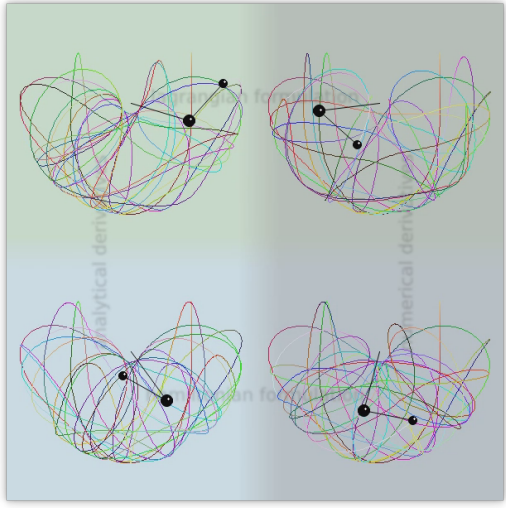
Professional tables: environmentally-assisted fatigue

Pair A	Pair B	Applied Cycles A	Applied Cycles B	M+B STRESS (psi)	K_e	Total Stress (psi)	S_{alt} (psi)	N_a	n_a	U_a	Max. Metal Temp. (°F)	DO (ppm)
694	447	5	20	125542.9	2.580	144164.4	220490.4	140.005	5	0.0357	566.6	0.150
699	447	50	15	121622.8	2.405	139047.0	198300.6	178.958	15	0.0838	566.6	0.550
699	1021	35	20	104691.5	1.653	126037.5	124507.0	582.468	20	0.0343	600.4	0.550
699	899	15	50	89695.4	1.000	102302.8	57864.5	6339.47	15	0.0024	336.1	0.550
695	899	5	35	84993.9	1.000	98798.6	55882.4	7027.83	5	0.0007	336.1	0.550
185	899	20	30	68222.2	1.000	76465.1	43250.2	15549.1	20	0.0013	336.1	0.550
1432	899	20	10	66665.7	1.000	83098.8	47002.3	11892.7	10	0.0008	336.1	0.550
1432	1653	10	100	49437.0	1.000	61950.9	33687.5	35734.8	10	0.0003	103.0	0.522
1296	1653	20	90	32478.6	1.000	38719.1	22025.4	154852	20	0.0001	368.2	0.522
1136	1653	20	70	27045.6	1.000	33751.1	19388.7	258499	20	0.0001	417.7	0.522
2215	1653	100	50	25255.9	1.000	25668.1	15147.6	1.15E+06	50	0.0000	547.0	0.522
2215	1213	50	20	22343.7	1.000	25298.3	14929.4	1.30E+06	20	0.0000	547.0	0.050
2215	1562	30	20	22047.7	1.000	24970.1	14735.7	1.46E+06	20	0.0000	547.0	0.050
2215	1	10	20	11956.0	1.000	12255.6	7232.5	1.00E+11	10	0.0000	547.0	0.150
1347	1	20	10	3786.5	1.000	4173.0	2412.1	1.00E+11	10	0.0000	450.0	0.150
1347	1595	10	20	3408.0	1.000	3430.2	1963.3	1.00E+11	10	0.0000	398.7	0.050
960	1595	20	10	241.8	1.000	259.9	146.0	1.00E+11	10	0.0000	299.5	0.050
960	960	5	5	0.0	1.000	0.0	0.0	1.00E+11	10	0.0000	299.5	0.050
										TOTAL CUF =	0.1596	

j	A_j	B_j	$n(A_j)$	$n(B_j)$	MB'_j [ksi]	$k_{e,j}$	S'_j [ksi]	$S_{alt,j}$ [ksi]	N_j	n_j	U_j	$T_{max,j}$ [°F]
1	447	694	20	5	125.5	2.580	144.2	220.400	1.40×10^2	5	3.57×10^{-2}	566.6
2	447	699	15	50	121.6	2.405	139	198.300	1.79×10^2	15	8.38×10^{-2}	566.6
3	699	1020	35	20	104.7	1.653	126.5	124.900	5.77×10^2	20	3.47×10^{-2}	599.2
4	699	899	15	50	89.7	1.000	102.3	62.640	5.02×10^3	15	2.99×10^{-3}	336.1
5	695	899	5	35	84.99	1.000	98.8	59.750	5.77×10^3	5	8.67×10^{-4}	336.1
6	899	1432	30	20	66.67	1.000	83.1	50.360	9.56×10^3	20	2.09×10^{-3}	634.2
7	184	899	20	10	68.23	1.000	76.76	46.440	1.24×10^4	10	8.09×10^{-4}	600.0
8	184	1641	10	100	51.22	1.000	55.83	33.630	3.59×10^4	10	2.78×10^{-4}	634.2
9	1296	1641	20	90	32.69	1.000	38.94	22.110	1.53×10^5	20	1.31×10^{-4}	366.2
10	1134	1641	20	70	27.31	1.000	34.49	19.800	2.34×10^5	20	8.53×10^{-5}	419.2
11	1641	2215	50	100	25.47	1.000	25.89	15.270	1.07×10^6	50	4.66×10^{-5}	547.0
12	1213	2215	20	50	22.34	1.000	25.3	14.930	1.31×10^6	20	1.53×10^{-5}	547.0
13	1630	2215	100	30	24.88	1.000	25.2	14.870	1.35×10^6	30	2.22×10^{-5}	547.0
14	1347	1630	20	70	16.71	1.000	17.12	9.798	3.72×10^9	20	5.38×10^{-9}	398.7
15	960	1630	20	50	13.54	1.000	13.95	8.405	7.76×10^{10}	20	2.58×10^{-10}	634.2
16	1595	1630	20	30	13.3	1.000	13.69	7.690	1.00×10^{11}	20	2.00×10^{-10}	299.4
17	1	1630	20	10	12.92	1.000	12.95	7.469	1.00×10^{11}	10	1.00×10^{-10}	450.0
18	1	1596	10	100	12.92	1.000	12.95	7.469	1.00×10^{11}	10	1.00×10^{-10}	450.0
19	1562	1596	20	90	2.829	1.000	0.2345	0.132	1.00×10^{11}	20	2.00×10^{-10}	299.4
											CUF total =	0.1615

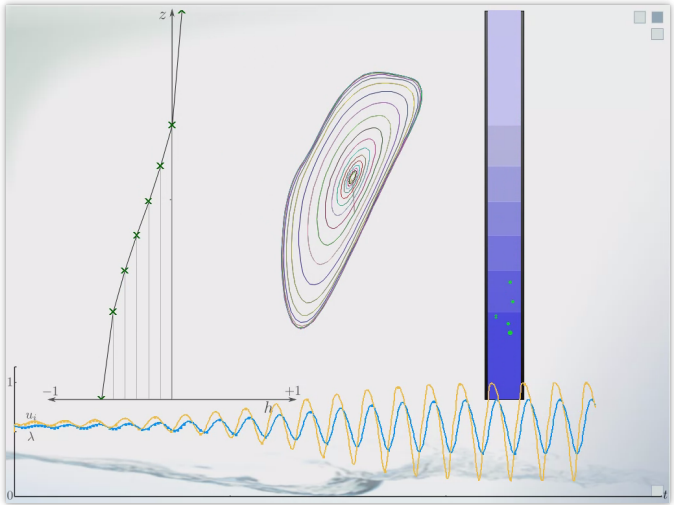
- Computation of NUREG-EPRI sample problem for Environmentally-assisted fatigue in NPP piping
- Top is a table from a publication by a multi-billion dollar agency
- Bottom is a PDF from FeenoX output piped through
  - AWK
  - LaTeX

Data for videos I: four double pendulums (v2)



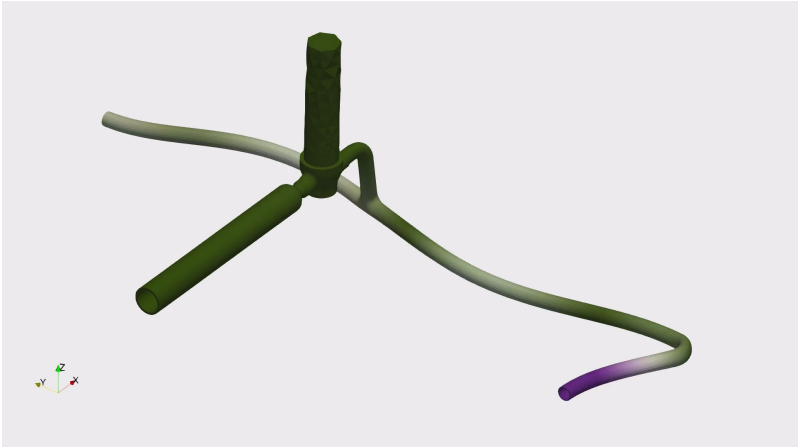
https://www.seamplex.com/feenox/videos/pendulums.webm

Data for videos II: boiling channel with sinusoidal power profile (v2)



https://www.seamplex.com/feenox/videos/sine.webm

Data for videos III: modal analysis for seismic analysis of piping (v2)



https://www.seamplex.com/feenox/videos/mode5.mp4
https://www.seamplex.com/feenox/videos/mode6.mp4
https://www.seamplex.com/feenox/videos/mode7.mp4

Complex figures: 2D IAEA PWR Benchmark (v2)

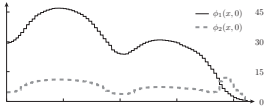
milonga's 2D LWR IAEA Benchmark Problem case #013
quarter-symmetry core meshed using delaunay (quads, ℓ_c =2) solved with finite volumes

largest eigenvalue k_{eff}	1.029927 (2905.71 pcm)
$\max \phi_2(x,y)$@core	11.31 @ (31.75, 29.99)
$\max \phi_2(x,y)$@reflector	12.20 @ (130.97, 50.98)
number of unknowns	15576
outer iterations	3
linear iterations	32
inner iterations	1947
residual norm	1.394×10^{-8}
relative error	6.865×10^{-9}
error estimate	3.425×10^{-9}
memory used	56256 kB
soft page faults	15645
hard page faults	0
total CPU time	0.9201 seconds

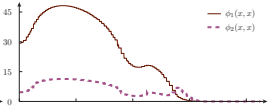
k	P_k	ϕ_{1k}	ϕ_{2k}
1	0.74	32.44	5.49
2	1.33	42.30	9.82
3	1.47	46.51	10.92
4	1.24	39.46	9.17
5	0.60	26.45	4.43
6	0.94	29.97	6.96
7	0.93	29.38	6.91
8	0.74	20.05	5.45
9	—	3.01	7.46
10	1.45	45.88	10.76
11	1.50	47.20	11.08
12	1.33	41.98	9.84
13	1.09	34.65	8.05
14	1.04	32.80	7.69
15	0.95	29.86	7.04
16	0.71	19.62	5.28
17	—	2.93	7.15
18	1.48	46.80	10.98
19	1.36	42.88	10.07
20	1.19	37.55	8.81
21	1.07	33.85	7.96
22	0.97	29.05	7.21
23	0.67	16.53	4.93
24	—	2.20	5.56
25	1.21	38.08	8.93
26	0.98	31.15	7.24
27	0.92	28.89	6.80
28	0.82	22.47	6.15
29	—	5.43	12.41
30	—	0.67	2.60
31	0.45	10.77	3.30
32	0.69	20.76	5.09
33	0.58	14.60	4.28
34	—	2.26	5.74
35	0.56	13.64	4.13
36	—	3.71	8.21
37	—	0.52	2.00
38	—	0.60	2.38

(a) Mesh and thermal flux distribution

(b) Power and fluxes



(c) Flux distribution $\phi_g(x,0)$ along the x axis



(d) Flux distribution $\phi_g(x,x)$ along the diagonal

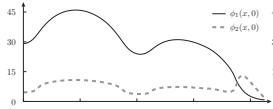
milonga's 2D LWR IAEA Benchmark Problem case #018
quarter-symmetry core meshed using delaunay (quads, ℓ_c =2) solved with finite elements

largest eigenvalue k_{eff}	1.029695 (2883.88 pcm)
$\max \phi_2(x,y)$@core	11.12 @ (30.76, 30.32)
$\max \phi_2(x,y)$@reflector	8.38 @ (50.00, 130.00)
number of unknowns	15922
outer iterations	3
linear iterations	32
inner iterations	1990
residual norm	1.056×10^{-8}
relative error	5.202×10^{-9}
error estimate	5.122×10^{-9}
memory used	109964 kB
soft page faults	30591
hard page faults	0
total CPU time	1.58 seconds

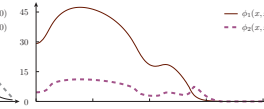
k	P_k	ϕ_{1k}	ϕ_{2k}
1	0.74	32.20	5.49
2	1.30	41.52	9.61
3	1.44	45.51	10.68
4	1.20	38.43	8.90
5	0.61	26.48	4.52
6	0.94	29.96	6.93
7	0.94	29.59	6.96
8	0.72	20.61	5.69
9	—	3.50	8.62
10	1.42	44.95	10.54
11	1.47	46.34	10.88
12	1.31	41.27	9.68
13	1.07	34.09	7.89
14	1.04	32.75	7.68
15	0.96	30.03	7.08
16	0.71	20.14	5.54
17	—	3.42	8.20
18	1.46	46.06	10.81
19	1.34	42.22	9.91
20	1.18	37.11	8.71
21	1.07	33.75	7.94
22	0.98	29.29	7.27
23	0.62	16.92	5.22
24	—	2.58	6.32
25	1.19	37.52	8.80
26	0.96	30.84	7.15
27	0.91	28.58	6.73
28	0.80	22.96	6.33
29	—	6.11	12.77
30	—	0.80	3.18
31	0.47	20.43	3.50
32	0.69	20.88	5.10
33	0.54	14.63	4.50
34	—	2.55	6.48
35	0.51	14.18	4.39
36	—	4.10	8.54
37	—	0.64	2.52
38	—	0.71	2.85

(a) Mesh and thermal flux distribution

(b) Power and fluxes



(c) Flux distribution $\phi_g(x,0)$ along the x axis



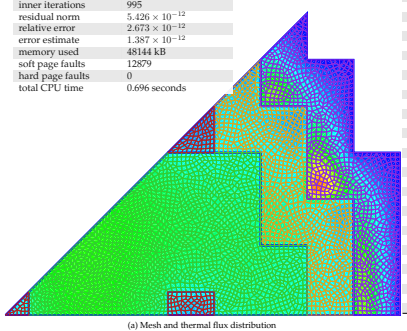
(d) Flux distribution $\phi_g(x,x)$ along the diagonal

Complex figures: 2D IAEA PWR Benchmark (v2)

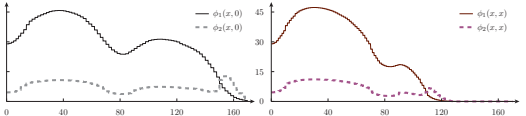
milonga's 2D LWR IAEA Benchmark Problem case #053
eighth-symmetry core meshed using delaunay (quads, $\ell_c = 2$) solved with finite volumes

largest eigenvalue k_{eff}	1.029966 (2009.42 pcm)
$\max \phi_2(x,y)$@core	11.10 @ (31.85, 30.64)
$\max \phi_2(x,y)$@reflector	11.70 @ (130.75, 50.81)
number of unknowns	7960
outer iterations	3
linear iterations	32
inner iterations	995
residual norm	5.426×10^{-12}
relative error	2.673×10^{-12}
error estimate	1.387×10^{-12}
memory used	48144 kB
soft page faults	12879
hard page faults	0
total CPU time	0.696 seconds

k	P_k	ϕ_{1k}	ϕ_{2k}
1	0.72	31.77	5.35
2	1.30	41.45	9.62
3	1.44	45.36	10.65
4	1.20	38.31	8.90
5	0.59	26.28	4.40
6	0.95	30.24	7.02
7	0.95	29.85	7.02
8	0.76	20.76	5.64
9	—	3.18	7.81
10	1.42	44.88	10.53
11	1.47	46.22	10.85
12	1.31	41.27	9.68
13	1.07	34.20	7.95
14	1.04	32.95	7.73
15	0.96	30.25	7.13
16	0.73	20.26	5.45
17	—	3.02	7.41
18	1.46	45.96	10.79
19	1.34	42.24	9.91
20	1.18	37.11	8.71
21	1.07	33.84	7.96
22	0.98	29.28	7.27
23	0.68	16.83	5.02
24	—	2.19	5.46
25	1.19	37.47	8.79
26	0.98	31.07	7.23
27	0.91	28.75	6.77
28	0.84	22.62	6.19
29	—	5.65	12.31
30	—	0.67	2.62
31	0.46	20.19	3.39
32	0.69	20.81	5.10
33	0.57	14.47	4.24
34	—	2.16	5.63
35	0.56	14.11	4.17
36	—	3.68	8.16
37	—	0.53	2.04
38	—	0.60	2.38



(a) Mesh and thermal flux distribution (b) Power and fluxes

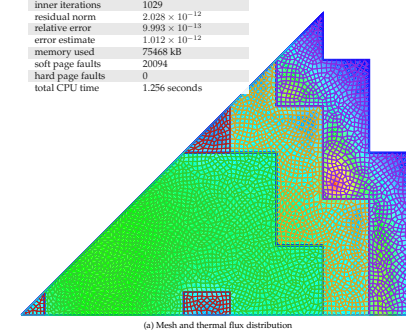


(c) Flux distribution $\phi_g(x,0)$ along the x axis (d) Flux distribution $\phi_g(x,x)$ along the diagonal

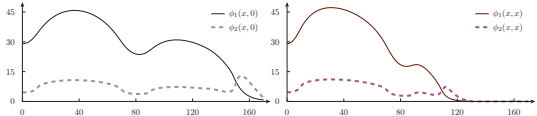
milonga's 2D LWR IAEA Benchmark Problem case #058
eighth-symmetry core meshed using delaunay (quads, $\ell_c = 2$) solved with finite elements

largest eigenvalue k_{eff}	1.029695 (2883.87 pcm)
$\max \phi_2(x,y)$@core	11.08 @ (30.45, 30.45)
$\max \phi_2(x,y)$@reflector	8.33 @ (130.00, 50.00)
number of unknowns	8234
outer iterations	3
linear iterations	32
inner iterations	1029
residual norm	2.028×10^{-12}
relative error	9.993×10^{-13}
error estimate	1.012×10^{-12}
memory used	75468 kB
soft page faults	20094
hard page faults	0
total CPU time	1.256 seconds

k	P_k	ϕ_{1k}	ϕ_{2k}
1	0.74	32.07	5.47
2	1.29	41.36	9.57
3	1.44	45.34	10.64
4	1.20	38.28	8.87
5	0.61	26.38	4.50
6	0.93	29.85	6.90
7	0.94	29.48	6.94
8	0.72	20.53	5.66
9	—	3.57	8.59
10	1.42	44.78	10.50
11	1.46	46.17	10.84
12	1.30	41.11	9.64
13	1.06	33.96	7.86
14	1.03	32.63	7.65
15	0.95	29.91	7.06
16	0.71	20.05	5.52
17	—	3.40	8.16
18	1.45	45.88	10.77
19	1.33	42.06	9.87
20	1.17	36.98	8.68
21	1.07	33.62	7.91
22	0.98	29.18	7.25
23	0.62	16.86	5.20
24	—	2.58	6.29
25	1.18	37.37	8.76
26	0.96	30.73	7.12
27	0.91	28.48	6.71
28	0.84	22.72	6.31
29	—	6.09	12.72
30	—	0.80	3.17
31	0.47	20.35	3.48
32	0.69	20.81	5.09
33	0.54	14.58	4.48
34	—	2.55	6.46
35	0.52	14.13	4.37
36	—	4.09	8.51
37	—	0.64	2.51
38	—	0.71	2.84

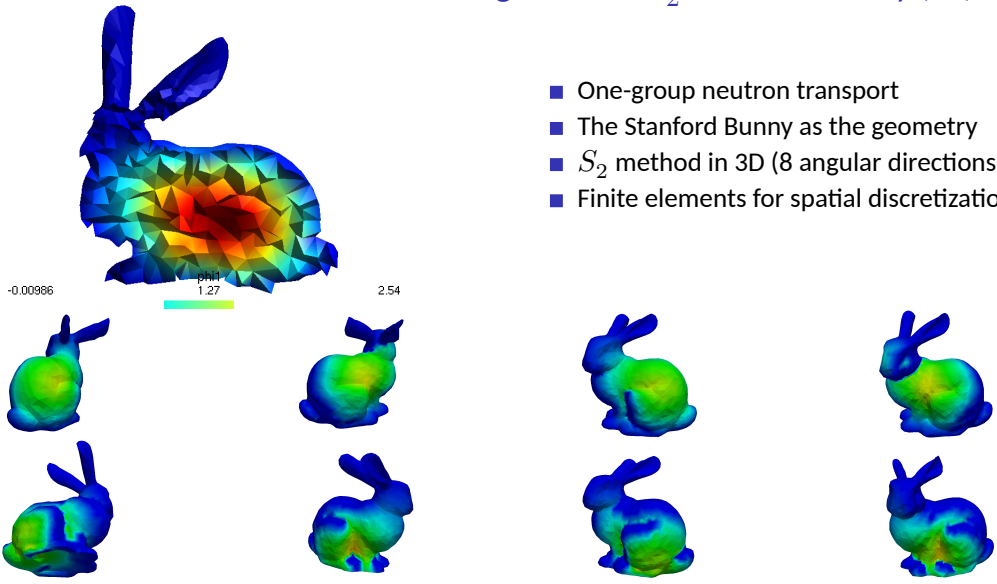


(a) Mesh and thermal flux distribution (b) Power and fluxes



(c) Flux distribution $\phi_g(x,0)$ along the x axis (d) Flux distribution $\phi_g(x,x)$ along the diagonal

Core-level neutronics over unstructured grids: the S_2 Stanford Bunny (v2)



- One-group neutron transport
- The Stanford Bunny as the geometry
- S_2 method in 3D (8 angular directions)
- Finite elements for spatial discretization

4. Quality Assurance

- Generic good software QA practices
  - Distributed version control system
  - Automated testing suites
  - User-reported bug tracking support
  - Signed releases
  - etc.

4.1. Reproducibility and traceability

- Both the source and the documentation should be tracked with a DVCS
- Repository should be accessible online
  - Might need credentials even for RO
- Version reporting
  - Executables must allow `--version`
  - Libraries must provide an API call
- The files needed to solve a problem should be simple & traceable by a DVCS

FeenoX

- Hosted on Github (`git`)
  - Previously on Bitbucket (`hg`)
  - Previously on Launchpad (`bzr`)
  - Previously on-premise (`svn`)
- https://github.com/seamplex/feenox
- https://seamplex.com/feenox
- Mailing list (Google group)
- Build a community! TO-DO
  - Code of conduct

```
$ feenox
FeenoX v0.1.12-gb9a534f-dirty
a free no-fee no-X uniX-like finite-element(ish) computational ↩
      engineering tool

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

- `-v`/`--version`: copyright notice
- `-V`/`--versions`: linked libraries
- Rule of generation: inputs from M4

4.2. Automated testing

- A mean to test the code is mandatory
- After each change
  - Check for regressions
  - Problems with already-computed solutions
  - Different from verification
- The compiler should not issue warnings
- Dynamic memory allocation checks are recommended
- Good practices are suggested
  - Unit testing
  - Continuous integration
  - Test coverage analysis

4.3. Bug reporting and tracking

- Users should be able to report bugs
  - A task should be created for each report
  - Address and document

FeenoX

- Standard test suite

```
$ make check
Making check in src
[...]
PASS: tests/trig.sh
PASS: tests/vector.sh
==========================================
Testsuite summary for feenox v0.1.12-gb9a534f
==========================================
# TOTAL: 26
# PASS:  25
# SKIP:  0
# XFAIL: 1
# FAIL:  0
# XPASS: 0
# ERROR: 0
==========================================
$
```

- Periodic `valgrind` runs
- Integration tests: TO-DO
- CI & test coverage: TO-DO

- Github issue tracker
- Branching & merging procedures: TO-DO

4.4 Verification

- Code must be always verified
- Check it solves **right the equations**
  - MES (mandatory)
  - MMS (recommended)
- One test case has to be added to the automated testing
- Third-party verification should be allowed
- Per-problem documentation

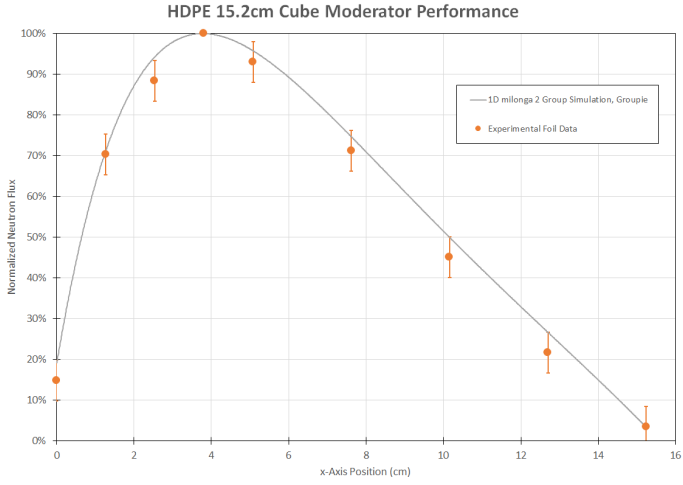
4.5. Validation

- Code should be validated as required
- Check it solves **the right equations**
  - Against experiments
  - Against other codes
- Third-party validation should be allowed
- Per-application/industry documentation
  - Procedures following standards

FeenoX

- There is a V&V report for the industrial human-less workflow project
  - Medical devices
  - Based on ASME V&V 40
- There is a lot to do!
- MES
  - Set of well-known benchmarks
  - NAFEMS, IAEA, etc.
- MMS
  - Everything is an expression
  - Parametric runs
  - `MESH_INTEGRATE` allows to compute L_2 norms directly in the `.fee`
- TL;DR: TO-DO

Experimental Validation



https://fusor.net/board/viewtopic.php?f=13&t=14087 ← because it is FOSS!

4.6. Documentation

- Documentation should be complete
  - User manual
    - Tutorial
    - Reference
  - Developer guide
- Quick reference cards, video tutorials, etc. not mandatory but recommended
- Non-trivial mathematics and methods
  - Explained
  - Documented
- Should be available as hard copies and mobile-friendly online
- Clear licensing scheme for the documentation
  - People extending the functionality ought to be able to document their work

FeenoX

- FeenoX is not compact!
  - Even I have to check the reference
- Commented sources: TO-DO
  - Keywords
  - Functions
  - Functionals
  - Variables
  - Material properties
  - Boundary conditions
  - Solutions
- Shape functions: TO-DO
- Gradient recovery: TO-DO
- Mathematical models: TO-DO

- Code is GPLv3+
- Documentation is GFDLv1.3+

Conclusions—FeenoX…

- closes a 15-year loop (2006–2021) with a third-system effect
- is to FEA what Markdown is to documentation
- is (so far) the only tool that fulfills 100% a fictitious SRS:
  - Free and open source (GPLv3+)
  - No recompilation needed
  - Cloud and web friendly
  - Human-less workflow
- follows the UNIX philosophy: “do one thing well”
- is already usable (and used!)
  - FeenoX v1.0 coincident with the PhD (TO-DO)
    - Laplace, heat, elasticity, modal, neutron transport & diffusion
    - Every current feature is there because there was at least one need from an actual project
  - Future versions online (TO-DO)
    - Electromagnetism? CFD? Schrödinger’s equation?
    - Python & Julia API
    - Coupled & multiphysics computations
    - Free and open-source online community