A free and open source computational tool for solving (nuclear-related) differential equations in the cloud

German (a.k.a. Jeremy) Theler

International Nuclear Atlantic Conference
Round Table 4 ENFIR "Best Estimate codes and methods
for Reactor Physics, Thermal Fluid Dynamics and Fuel Behavior"

3b3ae06—2021-12-01

FeenoX [1]

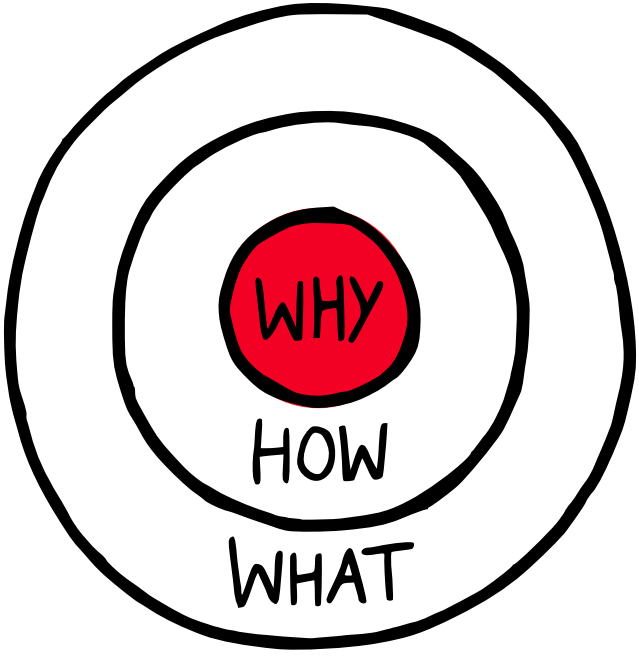
a free no-fee no-X uniX-like
finite-element(ish)
computational engineering tool

Presentation slides https://www.seamplex.com/feenox
~~Sources and~~ full examples at https://github.com/gtheler/2021-brasil

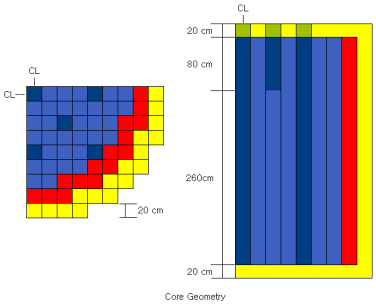
[1] FeenoX needs a logo.

Why → How → What

https://www.youtube.com/watch?v=u4ZoJKF_VuA



IAEA 2D/3D PWR benchmark (1976)



Two Group Cross Sections for Each Composition

Material		D_g	Σ_{rg}	$\nu\Sigma_{fg}$	$\Sigma_{g1\to2}$
	Fuel1	1,500	0,010	0,000	0,020
		0,400	0,085	0,135	
	Fuel1+Rod	1,500	0,010	0,000	0,020
		0,400	0,130	0,135	
	Fuel2	1,500	0,010	0,000	0,020
		0,400	0,080	0,135	
	Reflector	2,000	0,000	0,000	0,040
		0,300	0,010	0,000	
	Reflector+Rod	2,000	0,000	0,000	0,040
		0,300	0,055	0,000	

Figure 1: PARCS

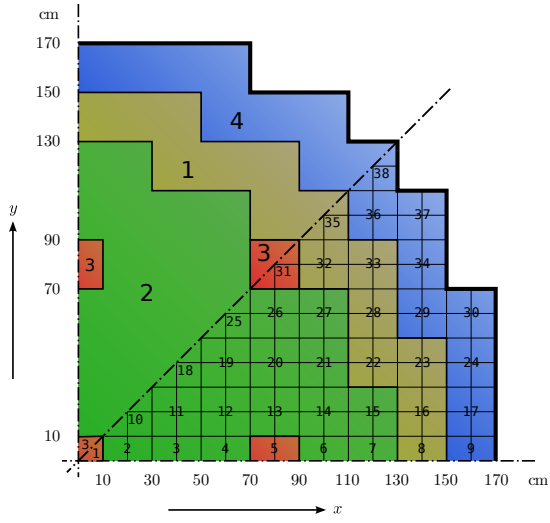
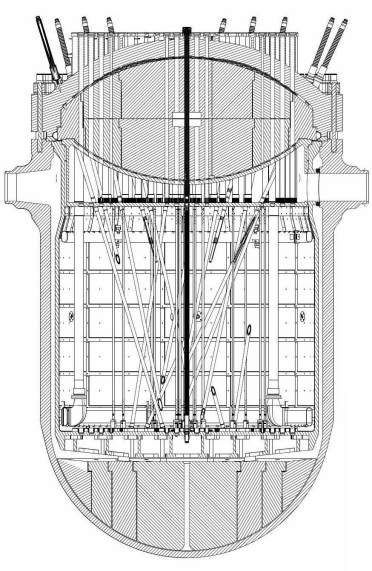
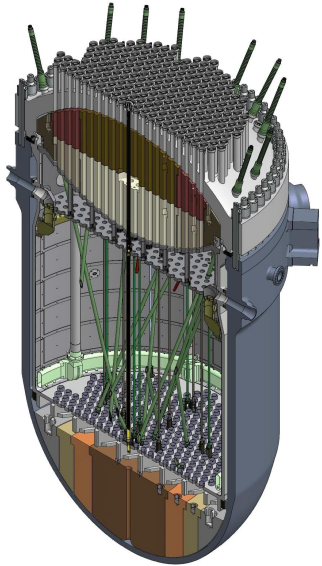


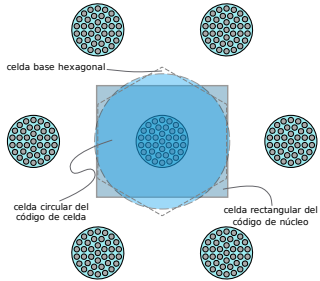
Figure 2: milonga

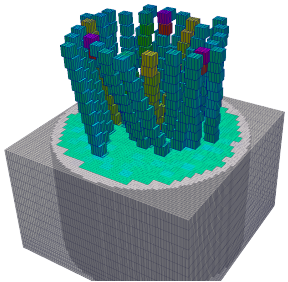
Vertical PHWR (Atucha)

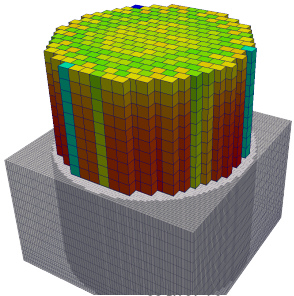


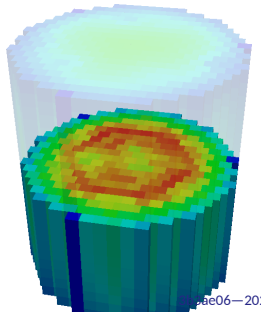


Steady-state core-level neutronic calculations

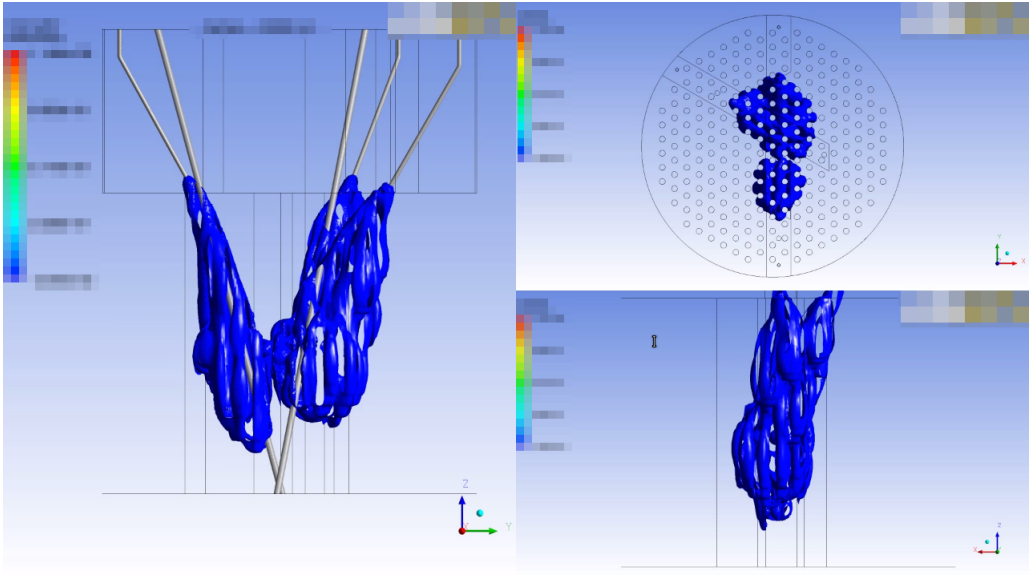




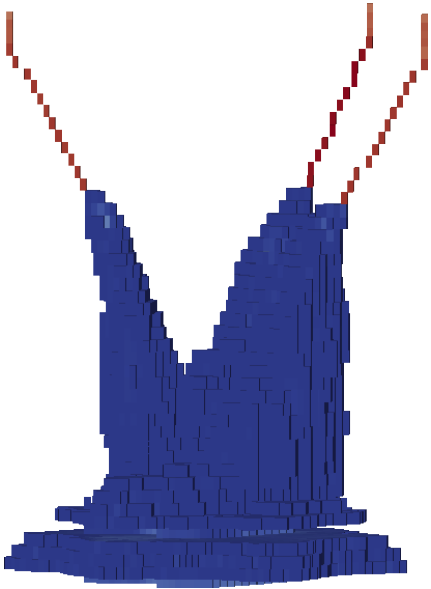


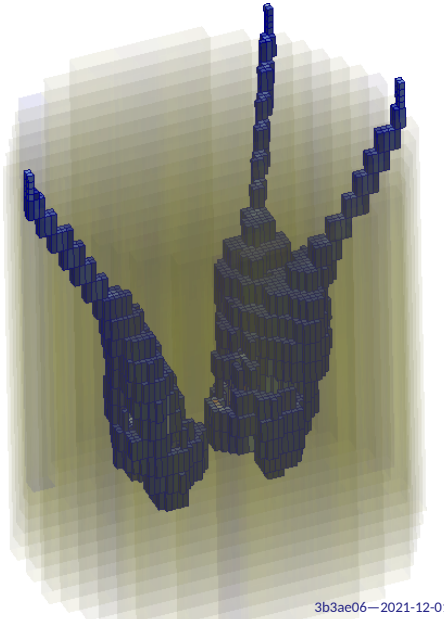


Transient emergency boron injection—CFD

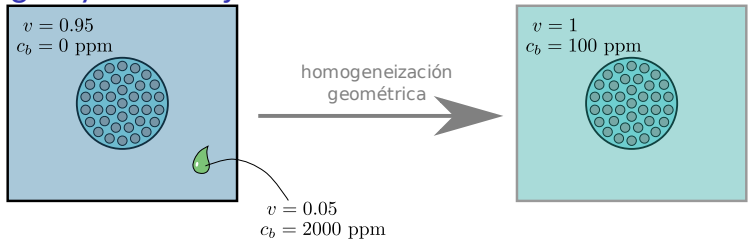


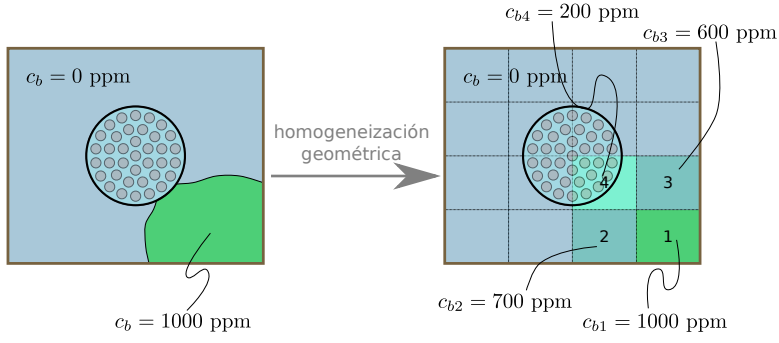
Transient emergency boron injection—core-level neutronics

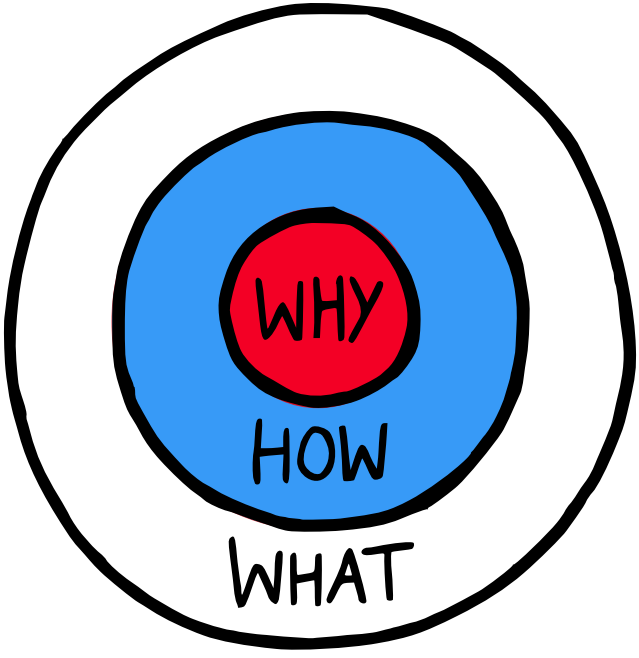




Transient emergency boron injection—cell-level neutronics







A little bit of history—College (2004–2008) (v1)

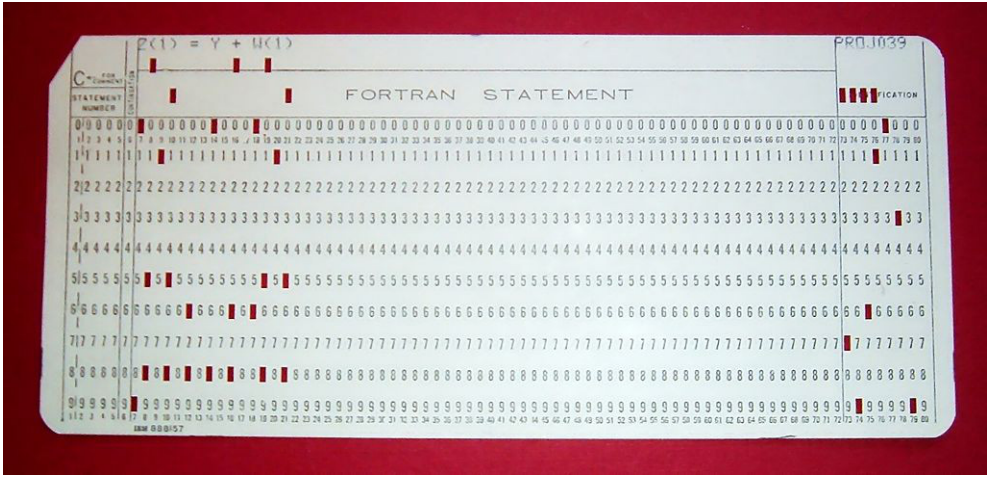
- Solve $\dot{\mathbf{x}} = F(\mathbf{x}, t)$
  a. program an *ad-hoc* numerical method
  b. use a standard numerical library in C or Python, or
  c. use a high-level system such as Octave, Maxima, etc.
- Point reactor equations solved “graphically” with non-free software

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

- I would rather write the equations in ASCII like

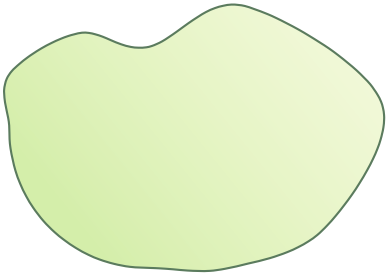
```
phi_dot = (rho-Beta)/Lambda * phi + sum(lambda[i], c[i], i, 1, N)
c_dot[i] = beta[i]/Lambda * phi - lambda[i]*c[i]
```

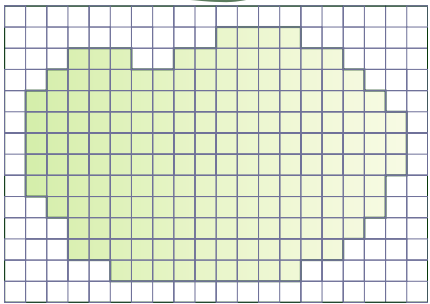
A little bit of history—Nuclear industry (2008–2014)

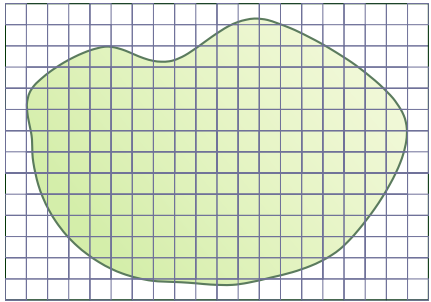


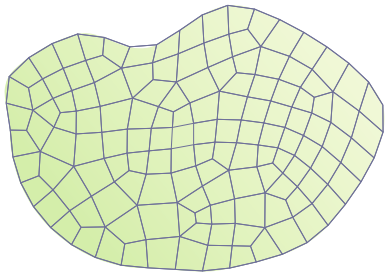
Modern/Advanced $\neq$ Fortran 90

Spatial discretizations

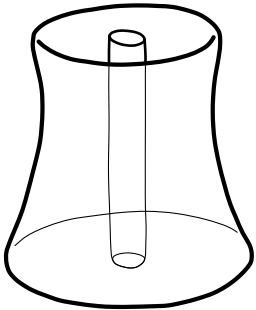




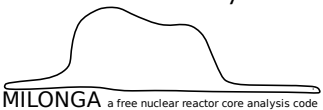




wasora & milonga (v2)



Wasora's an advanced suite for reactor analysis



- Coupled unstructured fine-mesh neutronics and thermal-hydraulics methodology using open software (2018)
- Open software one-step coupled neutronics and CFD thermalhydraulics calculation (2016)
- https://github.com/seamplex/milonga-2015-workshop
- Reactivity coefficient estimation by fuel temperature for Atucha II Nuclear Power Plant from neutron flux measurements (2014)
- On the design basis of a new core-level neutronic code written from scratch (2014)
- Neutron diffusion on unstructured grids: comparison between finite volumes and finite elements (2013)
- Geometric optimization of nuclear reactor cores (2013)
- Unstructured grids and the multigroup neutron diffusion equation (2013)

FeenoX (v3)

Software Requirement Specifications

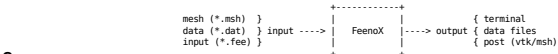
https://seamplex.com/feenox/doc/srs.html

- Industrial-level: open source for V&V
- Extensible: free (as in freedom)
- Cloud-first: programatically-defined
- Arbitrarily scalable for large problems
- Flexible
- Web & mobile-based GUIs
- QA
  - Reproducibility & traceability
  - Automated testing
  - Bug reporting & tracking
- V & V

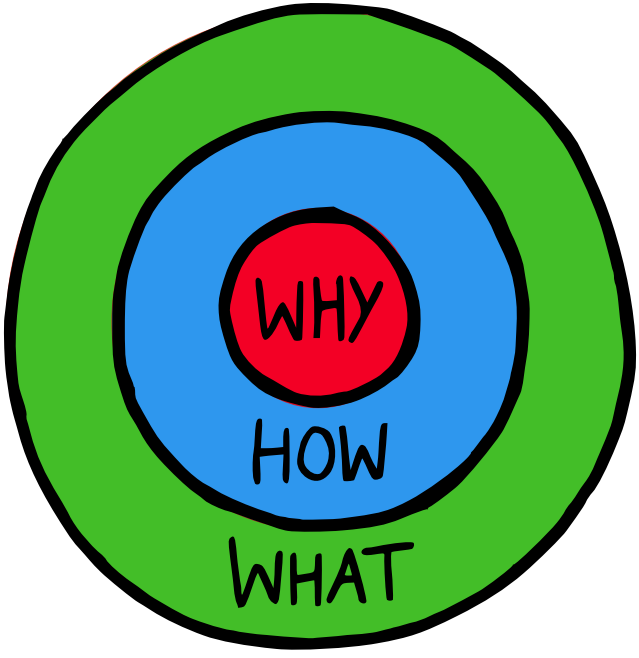
Software Design Specifications

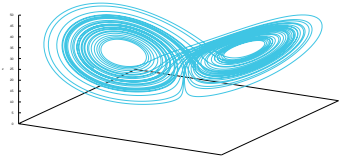
https://seamplex.com/feenox/doc/sds.html

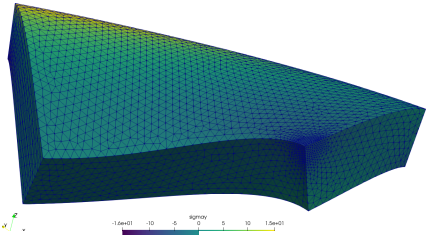
- GPLv3+: it is about *freedom* not price
- No-GUI script-friendly GNU/Linux binary

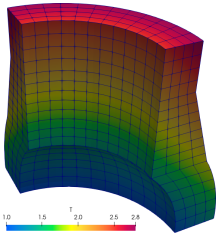


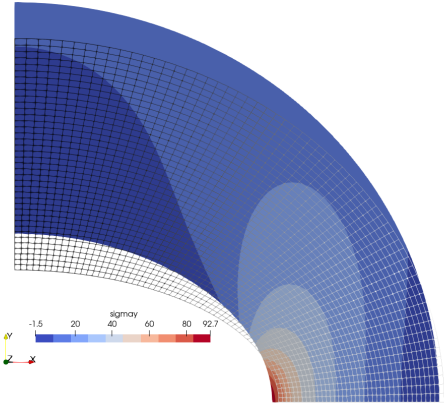
- Scalability based on UNIX
  - PETSc/SLEPc (MPI)
  - Gmsh (Metis)
- Flexibility shown in “what”
- Web GUI: https://www.caeplex.com
- Regression testing `make check`
- https://github.com/seamplex/feenox
- V & V: TODO! (help appreciated)!

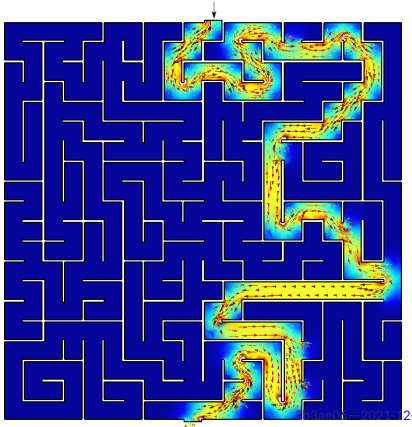





sigmay
-1.6e+01
-10
-5
0
5
10
1.5e+01


T
1.0
1.5
2.0
2.5
2.8


sigmay
-1.5
20
40
60
80
92.7



Point kinetics

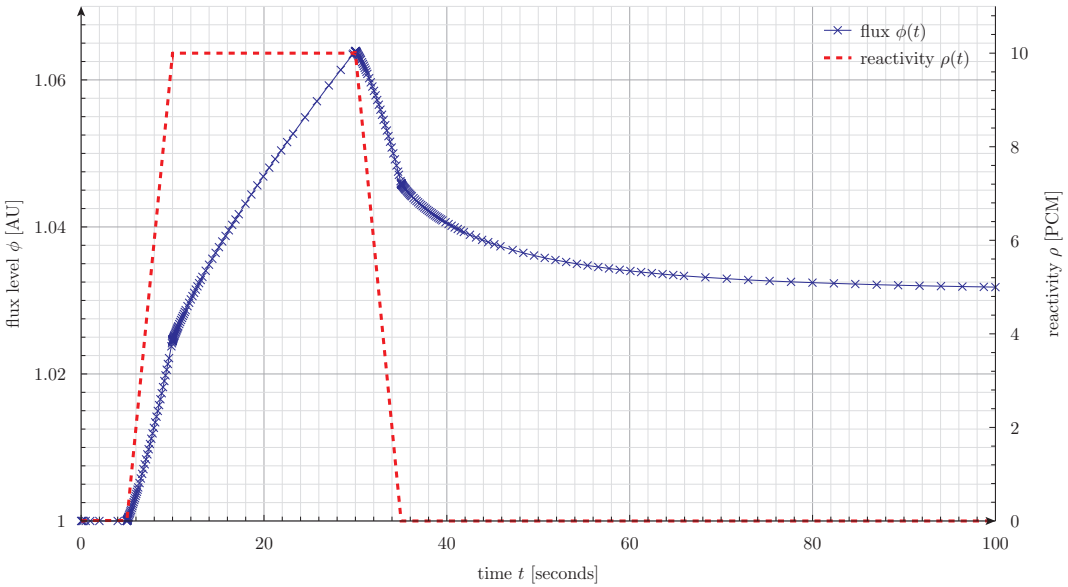
$$\begin{cases} \dot{\phi}(t) = \dfrac{\rho(t) - \mathrm{B}}{\Lambda} \cdot \phi(t) + \displaystyle\sum_{i=1}^{N} \lambda_i \cdot c_i \\ \dot{c}_i(t) = \dfrac{\beta_i}{\Lambda} \cdot \phi(t) - \lambda_i \cdot c_i \end{cases}$$

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
$
```

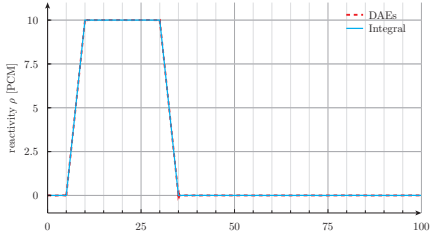

Point reactor equations with $\rho(t)$ given as scattered data
flux $\phi(t)$
reactivity $\rho(t)$
flux level ϕ [AU]
reactivity ρ [PCM]
time t [seconds]
1
1.02
1.04
1.06
0
2
4
6
8
10
0
20
40
60
80
100

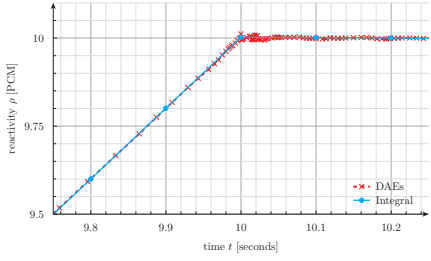
Inverse kinetics

```
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
```

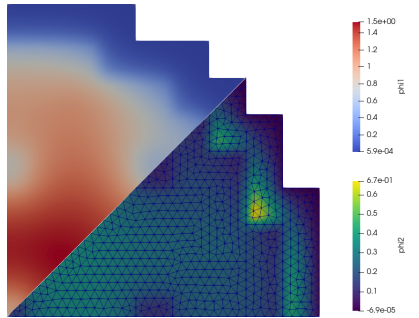




2D IAEA PWR Benchmark

```
PROBLEM neutron_diffusion 2D GROUPS 2
DEFAULT_ARGUMENT_VALUE 1 quarter   # either quarter or eigth
READ_MESH iaea-2dpwr-$1.msh

# each material is to a physical entity in the geometry file
Bg2 = 0.8e-4  # axial geometric buckling in the z direction
MATERIAL fuel1 {
  D1=1.5    Sigma_a1=0.010+D1(x,y)*Bg2    Sigma_s1.2=0.02
  D2=0.4    Sigma_a2=0.080+D2(x,y)*Bg2    nuSigma_f2=0.135  }

MATERIAL fuel2 {
  D1=1.5    Sigma_a1=0.010+D1(x,y)*Bg2    Sigma_s1.2=0.02
  D2=0.4    Sigma_a2=0.085+D2(x,y)*Bg2    nuSigma_f2=0.135  }

MATERIAL fuel2rod {
  D1=1.5    Sigma_a1=0.010+D1(x,y)*Bg2    Sigma_s1.2=0.02
  D2=0.4    Sigma_a2=0.130+D2(x,y)*Bg2    nuSigma_f2=0.135  }

MATERIAL reflector {
  D1=2.0    Sigma_a1=0.000+D1(x,y)*Bg2    Sigma_s1.2=0.04
  D2=0.3    Sigma_a2=0.010+D2(x,y)*Bg2 }

BC external vacuum=0.4692 # "external" is the name of the entity
BC mirror   mirror        # first is the name, second is the BC

SOLVE_PROBLEM                          # solve!
PRINT %.5f "keff = " keff              # print keff
WRITE_MESH iaea-2dpwr-$1.vtk phi1 phi2  # write fluxes
```

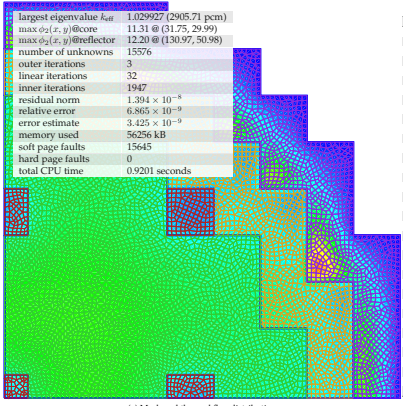
```
$ gmsh -2 iaea-2dpwr-quarter.geo
$ [...]
$ gmsh -2 iaea-2dpwr-eighth.geo
$ [...]
$ feenox iaea-2dpwr.fee quarter
keff =  1.02986
$ feenox iaea-2dpwr.fee eighth
keff =  1.02975
$
```



https://www.seamplex.com/feenox/examples/#iaea-2d-pwr-benchmark

2D IAEA PWR Benchmark

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

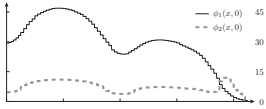


largest eigenvalue k_{eff}	1.029927 (2905.71 pcm)
$\max \phi_2(x,y)$@core	11.31 @ (31.75, 29.99)
$\max \phi_2(x,y)$@reflector	12.20 @ (130.97, 50.98)
number of unknowns	15576
outer iterations	3
linear iterations	32
inner iterations	1947
residual norm	1.394×10^{-8}
relative error	6.865×10^{-9}
error estimate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

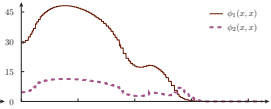
(a) Mesh and thermal flux distribution

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	0.83	22.47	6.15
29	—	5.43	12.41
30	—	0.67	2.60
31	0.45	19.77	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

(b) Power and fluxes

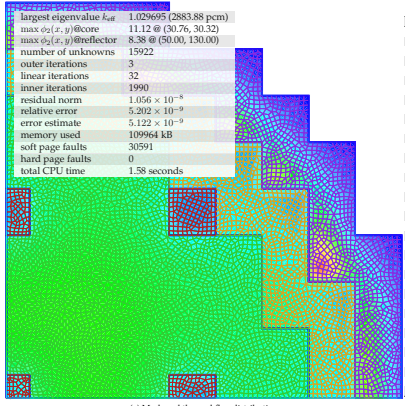


(c) Flux distribution $\phi_g(x,0)$ along the x axis



(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

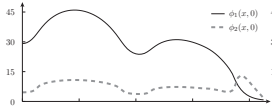


largest eigenvalue k_{eff}	1.029695 (2883.88 pcm)
$\max \phi_2(x,y)$@core	11.12 @ (30.76, 30.32)
$\max \phi_2(x,y)$@reflector	8.38 @ (50.00, 130.00)
number of unknowns	15922
outer iterations	3
linear iterations	32
inner iterations	1990
residual norm	1.056×10^{-8}
relative error	5.202×10^{-9}
error estimate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

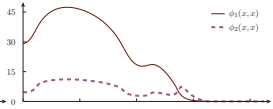
(a) Mesh and thermal flux distribution

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

(b) Power and fluxes



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



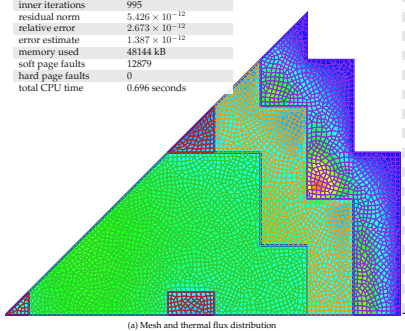
(d) Flux distribution $\phi_g(x,x)$ along the diagonal

2D IAEA PWR Benchmark

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

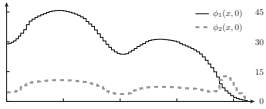
largest eigenvalue k_{eff}	1.029966 (2009.42 pcm)
$\max \phi_2(x,y)$@core	11.10 @ (31.85, 30.64)
$\max \phi_2(x,y)$@reflector	11.70 @ (130.75, 50.81)
number of unknowns	7960
outer iterations	3
linear iterations	32
inner iterations	995
residual norm	5.426×10^{-12}
relative error	2.673×10^{-12}
error estimate	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

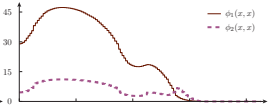


(a) Mesh and thermal flux distribution

(b) Power and fluxes



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

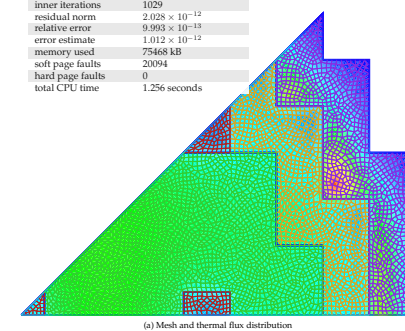


(d) Flux distribution $\phi_g(x,x)$ along the diagonal

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

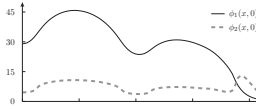
largest eigenvalue k_{eff}	1.029695 (2883.87 pcm)
$\max \phi_2(x,y)$@core	11.08 @ (30.45, 30.45)
$\max \phi_2(x,y)$@reflector	8.33 @ (130.00, 50.00)
number of unknowns	8234
outer iterations	3
linear iterations	32
inner iterations	1029
residual norm	2.028×10^{-12}
relative error	9.993×10^{-13}
error estimate	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

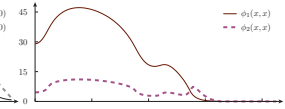


(a) Mesh and thermal flux distribution

(b) Power and fluxes

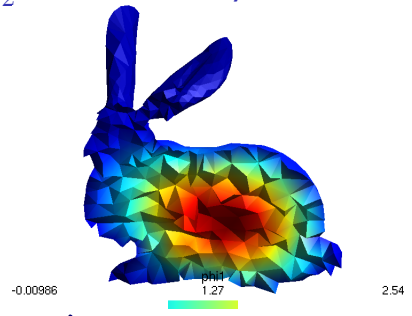


(c) Flux distribution $\phi_g(x,0)$ along the x axis



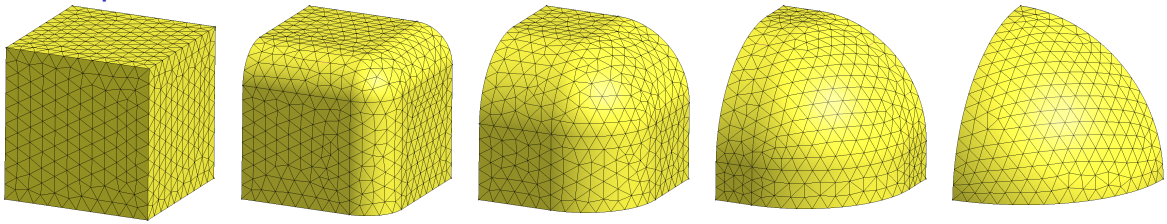
(d) Flux distribution $\phi_g(x,x)$ along the diagonal

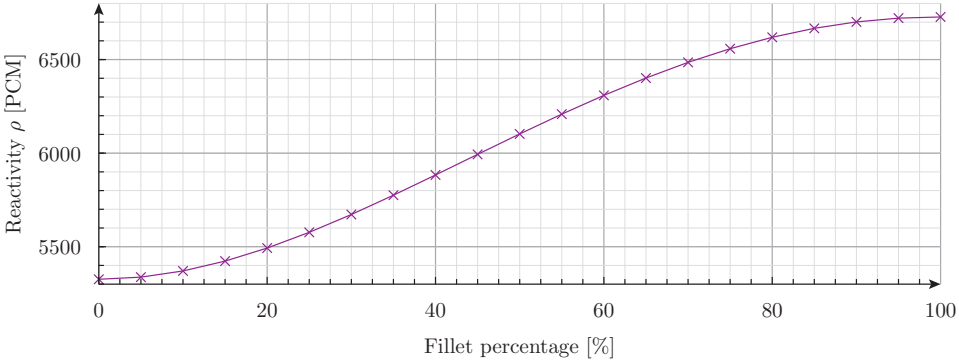
The S_2 Stanford Bunny



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

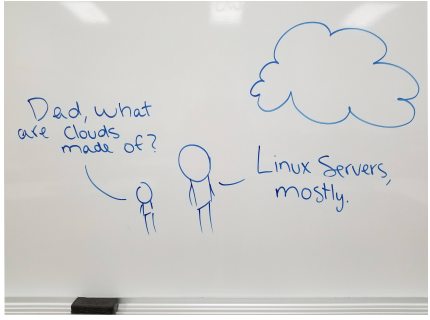
Cube-sphere





Conclusions & TODO

- 21st century: cloud-first approach
- Stick to free and open source software
  - free $\neq$ open source
- More formulations (milonga already had them)
  - FEM transport
  - FVM diffusion & transport
- Massive parallelization
  - MPI through PETSc/SLEPc
  - Metis through Gmsh
- Documentation
  - https://www.seamplex.com/feenox/doc
  - https://www.seamplex.com/feenox/examples
  - Come up with a nice logo
- Build a community!
  - https://github.com/seamplex/feenox/discussions



https://www.youtube.com/watch?v=Q-lKK4A2OzA