Fino

a free finite-element thermo-mechanical solver, v0.7

Jeremy Theler

April 16th, 2020

This manual is for fino (version v0.7, April 16th, 2020), which is a completely free-as-in-freedom finite-element thermo-mechancial solver desinged and implemented following the UNIX principles.

Table of Contents

Fino

1 Overview

Fino is a free and open source tool released under the terms of the GPLv3+ that uses the finite-element method to solve

- steady or quasistatic thermo-mechanical problems, or
- steady or transient heat conduction problems, or
- modal analysis problems.

Updates, examples, V&V cases and full reference: `https://www.seamplex.com/fino`

2 Running `fino`

2.1 Invocation

The format for running the `fino` program is:

```
fino [options] inputfile [optional_extra_arguments]...
```

The `fino` executable supports the following options:

`-d` or `--debug`
: Start in debug mode

`--node-debug`
: Ignore standard input, avoid debug mode

`-l` or `--list`
: List defined symbols and exit

`-h` or `--help`
: Display this help and exit

`-i` or `--info`
: Display detailed code information and exit

`-v` or `--version`
: Display version information and exit

`--mumps`
: use the MUMPS direct solver (if available)

`--progress`
: print ASCII progress bars for build, step and stress steps

`--petsc <option[=argument]>`
: Pass `-option argument` directly to PETSc/SLEPc, e.g.

```
$ fino tensile-test.fin --petsc ksp_view
```

The option `--petsc` is provided to avoid clashes with PETSc' non-POSIX arguments. Note that options are passed directly to PETSc/SLEPc if they do not clash with Fino/wasora. The same command as above could have been called as

```
$ fino tensile-test.fin -ksp_view
```

Input file instructions are read from standard input if a dash `-` is passed as `input-file`.

Fino accepts *optional extra arguments* which are then verbatimly replaced in the input file as `$1`, `$2`, and so on. So for example if an input file has a line like this

```
MESH_FILE $1.msh
[...]
```

Then two different meshes called `one.msh` and `two.msh` can successively be used in two runs with the same input file by calling Fino as

```
fino input.fin one
fino input.fin two
```

2.2 Example input files

2.2.1 Minimum working example

The following is a MWE input file for Fino that reads a Gmsh-generated `.msh` file, solves a linear elastic problem and wriets the results in a `.vtk` file which can be post-processed by Paraview:

```
MESH FILE_PATH tensile-test.msh  # mesh file in Gmsh format

E = 200e3   # [ MPa ] Young modulus ~ 200 GPa
nu = 0.3    # Poisson ratio

# boundary conditions ("left" and "right" come from the names in the mesh)
PHYSICAL_GROUP left  BC fixed
PHYSICAL_GROUP right BC Fx=1e4      # [ N ] load in x+

FINO_STEP   # solve

# write results (Von Mises, principal and displacements) in a VTK file
MESH_POST FILE_PATH tensile-mwe.vtk sigma sigma1 sigma2 sigma3 VECTOR u v w
```

2.2.2 Extended annotated example

The example can be extended to give more information as the following annotated input shows:

```
# tensile test example for Fino, see https://caeplex.com/p/41dd1
MESH FILE_PATH tensile-test.msh  # mesh file in Gmsh format (either version 2.2 or 4.x)

# uniform properties given as scalar variables
E = 200e3   # [ MPa ] Young modulus = 200 GPa
nu = 0.3    # Poisson's ratio

# boundary conditions ("left" and "right" come from the names in the mesh)
PHYSICAL_GROUP left  BC fixed       # fixed end
PHYSICAL_GROUP right BC Fx=1e4      # [ N ] load in x+

FINO_SOLVER PROGRESS_ASCII  # print ascii progress bars (optional)
FINO_STEP                   # solve

# compute reaction force at fixed end
FINO_REACTION PHYSICAL_GROUP left RESULT R

# write results (Von Mises, principal and displacements) in a VTK file
MESH_POST FILE_PATH tensile-test.vtk delta_sigma sigma sigma1 sigma2 sigma3 VECTOR u v w

# print some results (otherwise output will be null)
PRINT SEP " " "displ_max =" %.3f displ_max "mm"
PRINT SEP " " "sigma_max = (" %.1f sigma_max "±" delta_sigma_max ") MPa"
PRINT SEP " " "principal1 at center = (" %.5f sigma1(0,0,0) "±" delta_sigma(0,0,0) ") MPa"
PRINT SEP " " "reaction  = [" %.3e R "] Newtons"
PRINT FILE_PATH tensile-sigma.dat %.0f sigma(0,0,0)
```

3 Test case

This first case serves as a basic example to answer the first validation question: does Fino do what a FEA program is supposed to do? It also illustrates its design basis and the philosophy (https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html) behind its implementation. A quotation from Eric Raymond (http://www.catb.org/esr/)'s The Art of Unix Programming (http://www.catb.org/esr/writings/taoup/) helps to illustrate this idea:

> Doug McIlroy (https://en.wikipedia.org/wiki/Douglas_McIlroy), the inventor of Unix pipes (https://en.wikipedia.org/wiki/Pipeline_%28Unix%29) and one of the founders of the Unix tradition (https://en.wikipedia.org/wiki/Unix), had this to say at the time:
>
> 1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new features.
> 2. Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.
>
> [...]
>
> He later summarized it this way (quoted in "A Quarter Century of Unix" in 1994):
>
> - This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

Keep in mind that even though the quotes above and many FEA programs that are still mainstream today date both from the early 1970s, fifty years later they still

- Do not make just only one thing well.
- Do complicate old programs by adding new features.
- Do not expect the their output to become the input to another.
- Do clutter output with extraneous information.
- Do use stringently columnar and/or binary input (and output!) formats.
- Do insist on interactive output.

A further note is that not only is Fino both free (https://www.gnu.org/philosophy/free-sw.en.html) and open-source (https://opensource.com/resources/what-open-source) software but it also is designed to connect and to work with (rule of composition (https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2877684)) other free and open source software, like Gmsh (http://gmsh.info/), ParaView (https://www.paraview.org/), Gnuplot (http://gnuplot.info/), Pyxplot (http://www.pyxplot.org.uk/), Pandoc (https://pandoc.org/), TeX (https://tug.org/), and many others, including of course the operating system GNU (https://www.gnu.org/)/Linux (https://www.kernel.org/). In particular, this report has been created from scratch using free and open source software only.

Fino also makes use of high-quality free and open source mathematical libraries which contain numerical methods designed by mathematicians and programmed by professional programmers, such as GNU Scientific Library (`https://www.gnu.org/software/gsl/`), PETSc (`https://www.mcs.anl.gov/petsc/`), SLEPc (`http://slepc.upv.es/`) (optional) and all its respective dependencies. This way, Fino bounds its scope to do only one thing and to do it well: to build and solve finite-element formulations of thermo-mechanical problems. And it does so on high grounds, both

1. ethical: since it is free software (`https://www.gnu.org/philosophy/open-source-misses-the-point.en.html`), all users can

   0. run,
   1. share,
   2. modify, and/or
   3. re-share their modifications.

   If a user cannot read or write code to make Fino suit her needs, at least she has the *freedom* to hire someone to do it for her, and

2. technological: since it is open source (`http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/`), advanced users can detect and correct bugs and even improve the algorithms. all bugs are shallow.

The reader is encouraged to consider and to evaluate the differences (both advantages and disadvantages) between the approach proposed in this work with traditional thermo-mechanical FEA software. The Git (`https://git-scm.com/`) repository containing Fino's source code can be found at `https://github.com/seamplex/fino`.

3.1 Problem description

A tensile test specimen of nominal cross-sectional area $A = 20$ mm $\times$ 5 mm $= 100$ mm^2 is fully fixed on one end (magenta surface) and a tensile load of $F_x = 10$ kN is applied at the other end (green surface). The Young modulus is $E = 200$ GPa and the Poisson's ratio is $\nu = 0.3$.

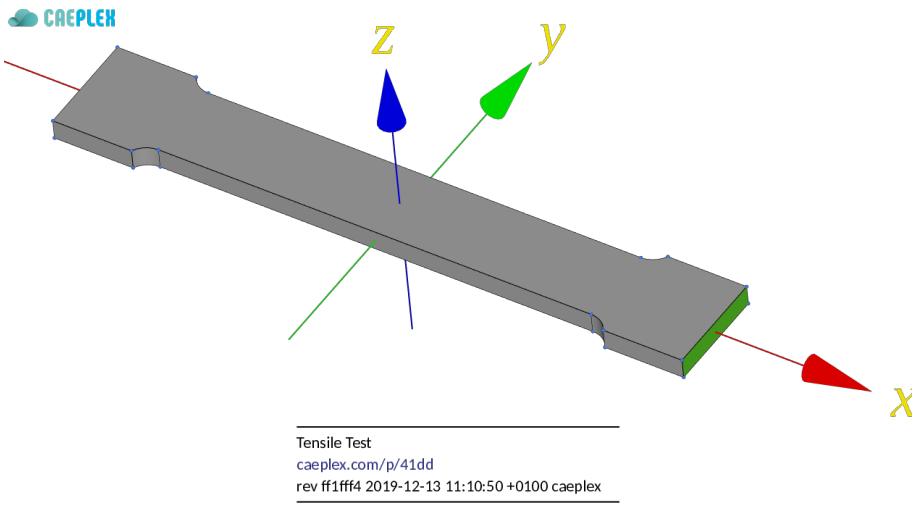


Figure 1: Tensile test specimen CAD from CAEplex `https://caeplex.com/p/41dd1`

3.1.1 Expected results

The displacements and stresses distribution within the geometry are to be obtained. Elongation along the x axis and a mild contraction in y (and even milder in z) are expected. The normal tension at the center of the specimen is to be checked to the theoretical solution $\sigma_x = F_x/A$ and the reaction at the fixed end should balance the external load $\vec{F}$ at the opposite face. Stress concentrations are expected to occur at sharp corners of the coupon.

3.2 Geometry and mesh

Following the general rule of performing only one thing well, and the particular rules of composition (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2877684`) and parsimony (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2878022`), the generation of the set of nodes and elements required to perform a thermo-mechanical computation using the finite element method is outside of Fino's scope. The finite-element mesh is an *input* to Fino.

In the particular case of the tensile test problem, the geometry is given as a STEP file (`tensile-test-specimen.step`). It is meshed by Gmsh (`http://gmsh.info/`) (or, following the rule of diversity (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2879078`), any other meshing tool which can write meshes in MSH format (`http://gmsh.info/doc/texinfo/gmsh.html#MSH-file-format`) keeping information about physical groups (`http://gmsh.info/doc/texinfo/gmsh.html#Elementary-entities-vs-physical-groups`)). A suitable mesh (fig. 2) can be created using the following `tensile-test.geo` file:

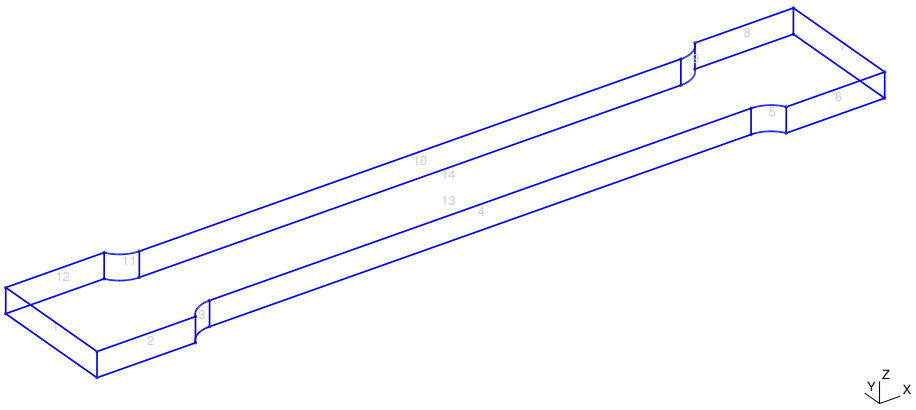
```
Merge "tensile-test-specimen.step"; // read the step file
Mesh.CharacteristicLengthMax = 1.5; // set the max element size lc
Mesh.ElementOrder = 2;              // ask for second-order elements

// define physical groups for BCs and materials
// the name in the LHS has to appear in the Fino input
```

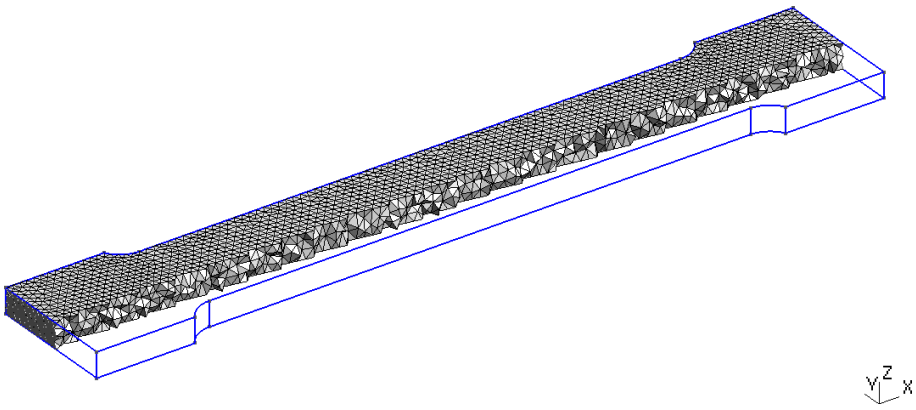
```
// the number in the RHS is the numerical id of the entity in the CAD file
Physical Surface ("left") =  {1};   // left face, to be fixed
Physical Surface ("right") =  {7};  // right face, will hold the load
Physical Volume ("bulk") =  {1};    // bulk material elements
```

Out of the six lines, the first three are used to read the CAD file, to set the characteristic element size $\ell_c = 1.5$ mm and to ask for second-order 10-noded tetrahedra (by default Gmsh creates first-order 4-node tetrahedra). The last three lines define one physical group each:

- geometrical surface #1 as physical surface "left," which will be set as fixed in the Fino input file,
- geometrical surface #7 as physical surface "right," which will hold the load defined in the Fino input file, and
- the volumetric bulk material elements in geometrical volume #1.



a



b

Figure 2: Tensile test specimen CAD, its geometrical entities and the resulting mesh.. a — Numerical ids of the surfaces in the original CAD., b — Three-dimensional mesh with uniform $\ell_c = 1.5$ mm and $\sim$ 50k nodes. In general, multi-solid problems need to have different physical volumes in order to Fino to be able to set different mechanical properties.

Even though this simple problem has a single solid, a physical volumetric group is needed in order to Gmsh to write the volumetric elements (i.e. tetrahedra) in the output MSH file (`tensile-test.msh`).

The usage of physical groups (`http://gmsh.info/doc/texinfo/gmsh.html#Elementary-entities-vs-physical-groups`) to define boundary conditions follows the rule of representation (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2878263`) as it folds knowledge into data instead of focusing on algorithmically setting loads on individual nodes. It also allows for extensibility (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2879112`) since, for example, a mesh with many physical groups can be used for both a tensile and a bending cases where the first uses some groups and the latter other groups bringing clarity (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2877610`) to the game.

3.3 Input file

Fino reads a plain-text input file—which in turns also reads the mesh generated above—that defines the problem, asks Fino to solve it and writes whatever output is needed. It is a syntactically-sweetened (`http://en.wikipedia.org/wiki/Syntactic_sugar`) way to ask the computer to perform the actual computation (which is what computers do). This input file, as illustrated in the example below lives somewhere near the English language so a person can read through it from the top down to the bottom and more or less understand what is going on (rule of least surprise (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2878339`)). Yet in the extreme case that the complexity of the problem asks for, the input file could be machine-generated by a script or a macro (rule of generation (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2878742`)). Or if the circumstances call for an actual graphical interface for properly processing (both pre and post) the problem, the input file could be created by a separate cooperating front-end such as CAEplex (`https://www.caeplex.com`) in fig. 1 above (rule of separation (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2877777`)). In any case, the input files—both for Gmsh and for Fino—can be tracked with Git (`https://en.wikipedia.org/wiki/Git`) in order to increase traceability and repeatability of engineering computations. This is not true for most of the other FEA tools avaialable today, particularly the ones that do not follow McIlroy's recommendations above.

Given that the problem is relatively simple, following the rule of simplicity (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2877917`), the input file `tensile-test.fin` (`tensile-test.fin`) ought to be also simple. Other cases with more complexity such as parametric runs (such as case_link(085-cantilever-cylinder)) or those that need to read results from other programs (such as case_link(075-fixed-compressed-cylinder)) in order to compare results might lead to more complex input files.

```
# tensile test example for Fino, see https://caeplex.com/p/41dd1
MESH FILE_PATH tensile-test.msh  # mesh file in Gmsh format (either version 2.2 or 4.x)

# uniform properties given as scalar variables
E = 200e3   # [ MPa ] Young modulus = 200 GPa
nu = 0.3    # Poisson's ratio
```

```
# boundary conditions ("left" and "right" come from the names in the mesh)
PHYSICAL_GROUP left  BC fixed       # fixed end
PHYSICAL_GROUP right BC Fx=1e4      # [ N ] load in x+

FINO_SOLVER PROGRESS_ASCII  # print ascii progress bars (optional)
FINO_STEP                   # solve

# compute reaction force at fixed end
FINO_REACTION PHYSICAL_GROUP left RESULT R

# write results (Von Mises, principal and displacements) in a VTK file
MESH_POST FILE_PATH tensile-test.vtk sigma sigma1 sigma2 sigma3 VECTOR u v w

# print some results (otherwise output will be null)
PRINT "displ_max = " %.3f displ_max "mm"
PRINT "sigma_max = " %.1f sigma_max "MPa"
PRINT "principal1 at center = " %.8f sigma1(0,0,0) "MPa"
PRINT "reaction  = [" %.3e R "] Newtons"
PRINT FILE_PATH tensile-sigma.dat %.0f sigma(0,0,0)
```

- The mesh `tensile-test.msh` (`tensile-test . msh`) is the output of Gmsh when invoked with the input `tensile-test.geo` (`tensile-test . geo`) above. It can be either version 4.1 (`http: / / gmsh . info / doc / texinfo / gmsh . html # MSH-file-format`) or 2.2 (`http: / / gmsh . info / doc / texinfo / gmsh . html # MSH-file-format-version-2-_0028Legacy_0029`).
- The mechanical properties, namely the Young modulus E and the Poisson's ratio ν are uniform in space. Therefore, they can be simply set using special variables `E` and `nu`.
- Boundary conditions are set by referring to the physical surfaces defined in the mesh. The keyword `fixed` is a shortcut for setting the individual displacements in each direction `u=0`, `v=0` and `w=0`.
- An explicit location within the logical flow of the input file hast to be given where Fino ought to actually solve the problem with the keyword `FINO_STEP`. It should be after defining the material properties and the boundary conditions and before computing secondary results (such as the reactions) and asking for outputs.
- The reaction in the physical group "left" is computed after the problem is solved (i.e. after `FINO_STEP`) and the result is stored in a vector named `R` of size three. There is nothing special about the name `R`, it could have been any other valid identifier name.
- A post-processing output file (`tensile-test . vtk`) in format VTK (`https: / / lorensen.github.io/VTKExamples/site/VTKFileFormats/`) is created, containing:
  - The von Mises stress `sigma` (σ) as an scalar field
  - The three principal stresses `sigma1`, `sigma_2` and `sigma_3` (σ_1, σ_2 and σ_3 respectively) as three scalar fields
  - The displacement vector $\vec{u} = [u, v, w]$ as a three-dimensional vector field
- Some results are printed to the terminal (i.e. the standard output (`https: / / en . wikipedia.org/wiki/Standard_streams#Standard_output_(stdout)`)) to summarize the run. Note that
  1. The actual output (including post-processing files) is 100% defined by the user, and

2. If no output instructions are given in the input file (`PRINT`, `MESH_POST`, etc.) then no output will be obtained.

Not only do these two facts follow the rule of silence (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2878450`) but they also embrace the rule of economy (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2878666`): the time needed for the user to find and process a single result in a soup of megabytes of a cluttered output file far outweighs the cost of running a computation from scratch with the needed result as the only output.

- Finally, the von Mises stress $\sigma(0,0,0)$ evaluated at the origin is written to an ASCII (`https://en.wikipedia.org/wiki/ASCII`) file `tensile-sigma.dat` (`tensile-sigma.dat`) rounded to the nearest integer (in MPa). This is used to test the outcome of Fino's self-tests using the `make check` target (`https://www.gnu.org/software/make/manual/make.html#Standard-Targets`) in an automated way (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#id2878742`). Note that there is no need to have an actual node at $\vec{x} = (0,0,0)$ since Fino (actually wasora (`https://www.seamplex.com/wasora`)) can evaluate functions at any arbitrary point.

3.4 Execution

Here is a static mimic of a 22-second terminal session:

```
$ gmsh -3 tensile-test.geo
Info    : Running 'gmsh -3 tensile-test.geo' [Gmsh 4.5.2-git-2373007b0, 1 node, max. 1 thread]
Info    : Started on Wed Jan 29 11:07:04 2020
Info    : Reading 'tensile-test.geo'...
Info    : Reading 'tensile-test-specimen.step'...
Info    :  - Label 'Shapes/ASSEMBLY/=>[0:1:1:2]/Pad' (3D)
Info    :  - Color (0.8, 0.8, 0.8) (3D & Surfaces)
Info    : Done reading 'tensile-test-specimen.step'
Info    : Done reading 'tensile-test.geo'
Info    : Meshing 1D...
Info    : [  0 %] Meshing curve 1 (Line)
Info    : [ 10 %] Meshing curve 2 (Line)
Info    : [ 10 %] Meshing curve 3 (Line)
[...]
Info    : [100 %] Meshing surface 14 order 2
Info    : [100 %] Meshing volume 1 order 2
Info    : Surface mesh: worst distortion = 0.90913 (0 elements in ]0, 0.2]); worst gamma = 0.722061
Info    : Volume mesh: worst distortion = 0.824145 (0 elements in ]0, 0.2])
Info    : Done meshing order 2 (1.32521 s)
Info    : 49534 nodes 40321 elements
Info    : Writing 'tensile-test.msh'...
Info    : Done writing 'tensile-test.msh'
Info    : Stopped on Wed Jan 29 11:07:07 2020
$ fino tensile-test.fin
................................................................................................
------------------------------------------------------------------------------------------------
================================================================================================
displ_max =     0.076   mm
sigma_max =     160.2   MPa
principal1 at center =  99.99998119     MPa
reaction  = [   -1.000e+04      -1.693e-04      -1.114e-03      ] Newtons
$
```

- The three lines with the dots, dashes and double dashes are ASCII progress bars for the assembly of the stiffness matrix, the solution of the linear system and the computation of stresses, respectively. They are turned on with `PROGRESS_ASCII`.

- Almost any location within the input file where a numerical value is expected can be replaced by an algebraic expression, including standard functions like `log`, `exp`, `sin`, etc. See wasora's reference (`https://www.seamplex.com/wasora/reference.html#functions`) for details.

- Once again, if the `MESH_POST` and `PRINT` instructions were not included, there would not be any default output of the execution (rule of silence (`http://www.linfo.org/rule_of_silence.html`)). This should be emphasized over and over, as I have recently (i.e. thirteen years after the commercial introduction of smartphones) stumbled upon a the output file of a classical FEM program that seems to have been executed in 1970: paginated ASCII text ready to be fed to a matrix-doy printed containing all the possible numerical output because the CPU cost of re-running the case of course overwhelms the hourly rate of the engineer that hast to understand the results. For more than fifty years (and counting), McIlroy's second bullet above has been blatantly ignored.

- It has already been said that the output is 100% controlled by the user. Yet this fact includes not just what is written but also how: the precision of the printed results is controlled with `printf` format specifiers (`https://en.wikipedia.org/wiki/Printf_format_string`). Note the eight decimal positions in the evaluation of σ_1 at the origin, whilst the expected value was 100 MPa (the load is $F_x = 10^4$ N and the cross-sectional area is 100 mm^2).

- If available, the MUMPS Solver (`http://mumps-solver.org/`) direct solver can be used instead of the default GAMG-preconditioned GMRES itearative solver by passing the option `--mumps` in the command line. More on solvers in sec. 3.6.2.

3.5 Results

After the problem is solved and an appropriately-formatted output file is created, Fino's job is considered done. In this case, the post-processing file is written using `MESH_POST`. The VTK output (`tensile-test.vtk`) can be post-processed with the free and open source tool ParaView (`http://www.paraview.org/`) (or any other tool that reads VTK files (`https://lorensen.github.io/VTKExamples/site/VTKFileFormats/`)) such as Gmsh in post-processing mode (`http://gmsh.info/doc/texinfo/gmsh.html#Post_002dprocessing`)), as illustrated in fig. 3.

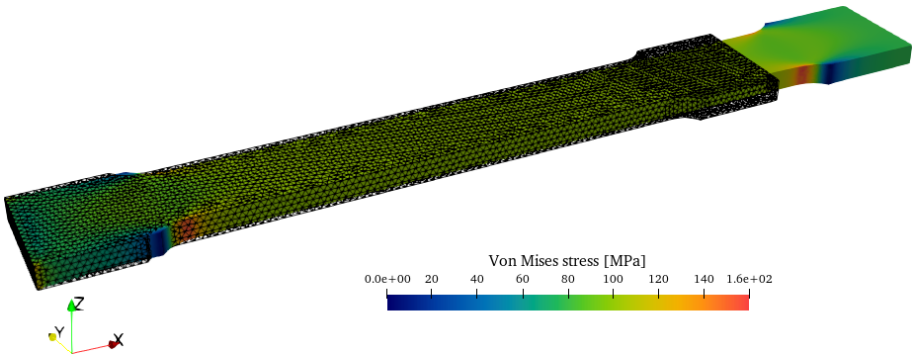


Figure 3: Tensile test results obtained by Fino and post-processed by ParaView. Displacements are warped 500 times.

3.5.1 Check

Qualitatively speaking, Fino does what a mechanical finite-element program is expected to do:

- The displacement vector and scalar von Mises stress fields are computed by Fino, as they are successfully read by Paraview.
- Elongation in the x direction and mild contractions in y and z are observed.
- The principal stress σ_1 should be equal to F_x/A, where
  - $F_x = 10^4$ N, and
  - $A = 20$ mm $\times$ 5 mm $= 100$ mm^2.

  In effect, $\sigma_1(0,0,0) = 100$ MPa.
- The numerical reaction $\vec{R}$ at the fixed end reported by Fino is
  $\vec{R} = [-10^4 \quad \approx 10^{-4} \quad \approx 10^{-3}]^T$ N
  which serves as a consistency check.
- Stress concentrations are obtained where they are expected.

3.6 Extra checks

The simple tensile test problem is qualitatively solved with Fino as expected. This section extends the validation to further check that Fino is solving the right equations.

3.6.1 Strain energy convergence

It is a well-known result from the mathematical theory that the displacement-based finite-element formulation gives a stiffer solution than the continuous problem. This means that the total strain energy U in load-driven (displacement-driven) problems is always lower (higher) than the exact physical value.

The following input files asks Fino to perform a parametric run on $c \in [1:10]$ which controls the characteristic element size $\ell_c = 10$ mm$/c$:

```
PARAMETRIC c MIN 1 MAX 10 STEP 1
```

```
lc = 10/c
M4 INPUT_FILE_PATH parametric.geo.m4 OUTPUT_FILE_PATH parametric.geo EXPAND lc
SHELL "gmsh -3 -v 0 parametric.geo"
MESH FILE_PATH parametric.msh
E = 200e3
nu = 0.3
PHYSICAL_GROUP left  BC fixed
INCLUDE $1.fin     # include either load or displ boundary condition
FINO_STEP
FINO_REACTION PHYSICAL_GROUP left RESULT R
PRINT c lc nodes %.4f strain_energy %.8f u(80,0,0) R(1) %.8f sigma1(0,0,0) %.3f time_wall_total %e memory
```

Depending on the command-line argument `$1`, it includes either `load.fin`

```
PHYSICAL_GROUP right BC Fx=1e4
```

or `displ.fin`

```
PHYSICAL_GROUP right BC u=0.075512349
```

```
$ fino parametric-energy.fin displ | tee displ.dat
10     10      1381    378.0254     0.07551236    -10012.19688820 100.12193130    0.393   5.705728e+07
20     5       3406    377.7806     0.07551234    -10005.87225589 100.05786431    1.416   1.020641e+08
30     3.33333 5934    377.7151     0.07551235    -10004.04718626 100.04035986    2.121   1.901896e+08
40     2.5     13173   377.6670     0.07551234    -10002.80454512 100.02703566    5.841   4.936827e+08
50     2       21897   377.6416     0.07551235    -10002.10898755 100.02189744    9.199   9.886843e+08
60     1.66667 36413   377.6316     0.07551234    -10001.84996627 100.01824358    20.676  1.319252e+09
70     1.42857 52883   377.6141     0.07551235    -10001.38248904 100.01358953    24.926  2.285650e+09
80     1.25    71568   377.6106     0.07551236    -10001.29858951 100.01271875    38.768  2.688750e+09
90     1.11111 103742  377.6026     0.07551235    -10001.09116871 100.01066954    67.782  3.595817e+09
100    1       136906  377.5994     0.07551234    -10000.99998694 100.00974332    90.248  4.684169e+09
$ fino parametric-energy.fin load | tee load.dat
10     10      1381    377.0997     0.07538952    -10000.00057800 99.99985582     0.826   5.935514e+07
20     5       3406    377.3440     0.07542967    -9999.99727028  99.99965822     2.036   1.042063e+08
30     3.33333 5934    377.4100     0.07544179    -10000.00700660 99.99980990     2.541   1.924178e+08
40     2.5     13173   377.4580     0.07545095    -9999.95509637  99.99981492     6.251   4.988027e+08
50     2       21897   377.4829     0.07545594    -9999.99976796  99.99974599     7.323   9.938166e+08
60     1.66667 36413   377.4938     0.07545809    -9999.97099411  99.99987546     14.838  1.328071e+09
70     1.42857 52883   377.5105     0.07546140    -9999.95702214  100.00028774    25.190  2.294686e+09
80     1.25    71568   377.5145     0.07546216    -9999.97300192  99.99997997     33.983  2.697781e+09
90     1.11111 103742  377.5201     0.07546350    -10000.04369761 99.99955431     45.707  3.604849e+09
100    1       136906  377.5267     0.07546443    -9999.93061385  100.00001476    99.697  4.685390e+09
$
```

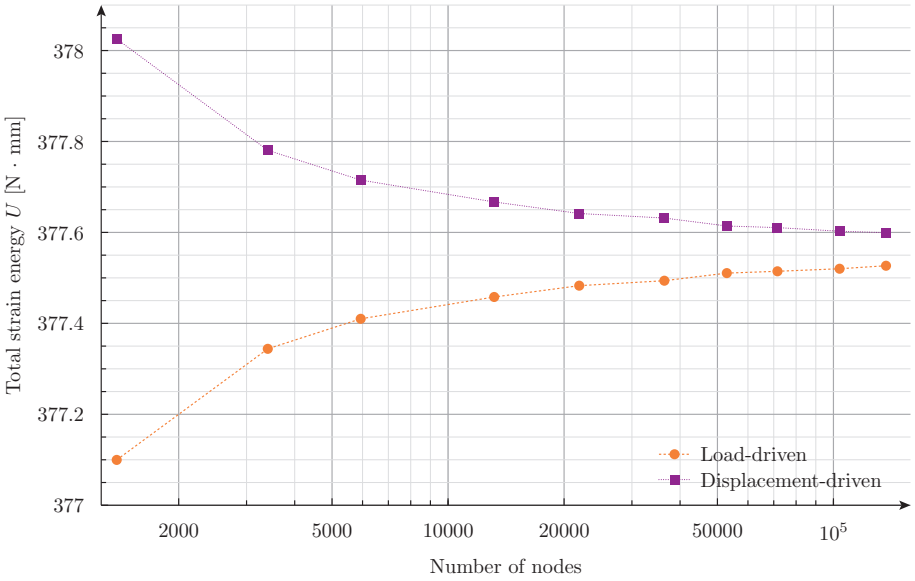


Figure 4: Total strain energy U computed by Fino as a function of the number of nodes for load and displacement-driven problems.

Indeed, fig. 4 shows that the total strain energy U is a monotonically increasing function of the number of nodes. Conversely, for the load-driven case it is monotonically decreasing, which is the expected behavior of a displacement-based finite-element program.

3.6.2 Performance

Let's switch our attention briefly to the subject of performance, which is indeed related to what it is expected from a finite-element program. In the general case, the time needed to solve a finite-element problem depends on

1. the size of the problem being solved,
   a. the number of the nodes in the mesh
   b. the number of degrees of freedom per node of the problem
2. the particular problem being solved,
   a. the condition number of the stiffness matrix
   b. the non-zero structure of the stiffness matrix
3. the computer used to solve the problem,
   a. the architecture, frequency and number of the CPU(s)
   b. the size, speed and number of memory cache levels
   c. implementation details of the operating-system scheduler
4. the optimization flags used to compile the code
5. the algorithms used to solve the system of equations
   a. preconditioner
   b. linear solver

c. parallelization (or lack of)

As it has been already explained, Fino uses PETSc (`https://www.mcs.anl.gov/petsc/`)—pronounced PET-see (the S is silent). It is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. In other words, it is a library programmed by professional programmers implementing state-of-the-art numerical methods developed by professional mathematicians. And yet, it is free and open source (`https://www.mcs.anl.gov/petsc/documentation/license.html`) software.

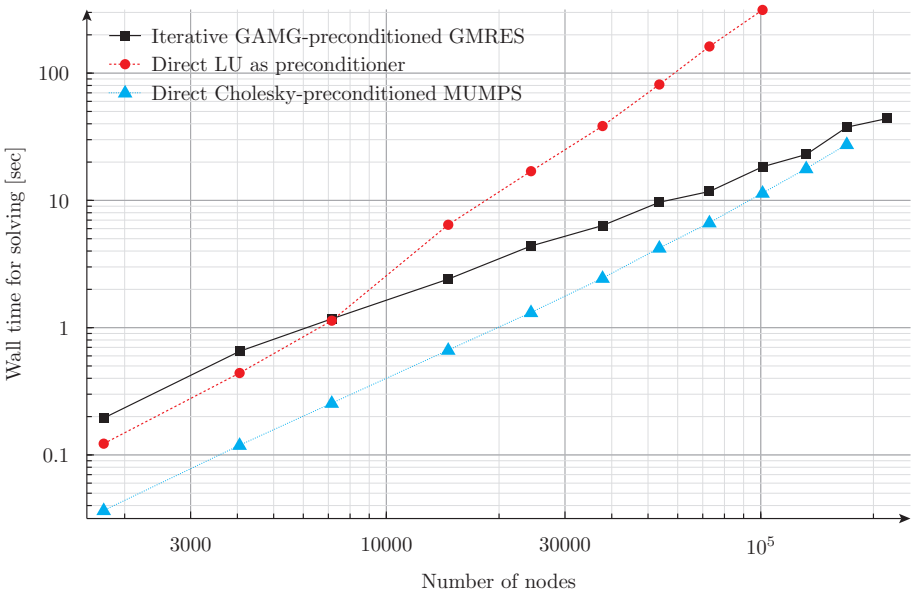
PETSc provides a variety of linear solvers and preconditioners (`https://www.mcs.anl.gov/petsc/documentation/linearsolvertable.html`) which can be used to solve the finite-element formulation. The choice of the type of preconditioner and linear solver can be done from the input file or directly from the command line. By default, mechanical problems are solved with the Geometric algebraic multigrid (`https://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/PC/PCGAMG.html`)-preconditioned Generalized Minimal Residual method (`https://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/KSP/KSPGMRES.html`), which is an iterative solver. An alternative might be a direct sparse solver, such as the MUMPS Solver (`http://mumps-solver.org/`), which Fino (through PETSc's interface) can use.

To get some insight about how the problem size, the computer and the algorithms impact in the time needed to solve the problem we perform another parametric run on $c \in [1:12]$ in two different computers with three different solvers and pre-conditioners:

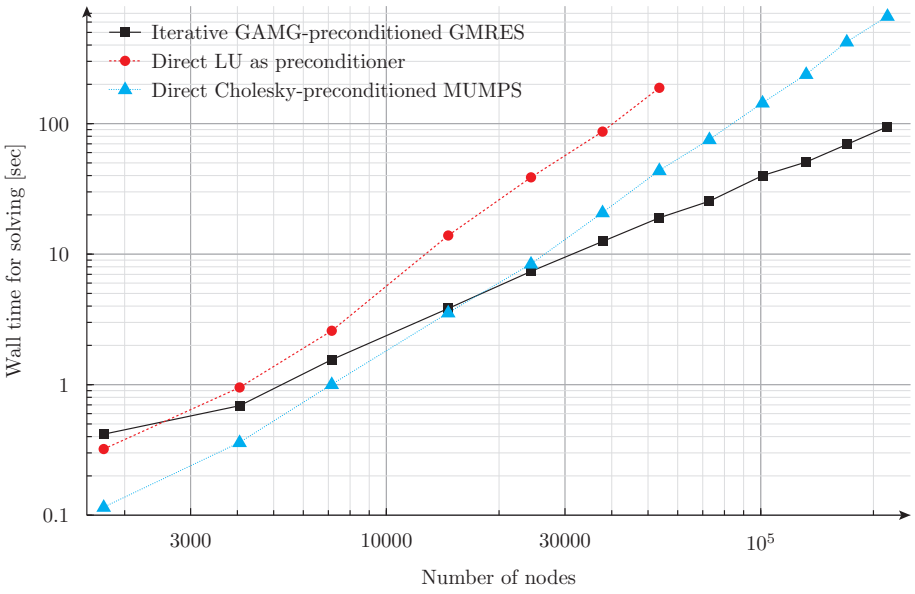
1. Geometric Algebraic Multigrid preconditioner with Generalized Minimal Residual solver (default)
2. LU direct solver used as a preconditoner
3. Cholesky-preconditioned direct MUMPS solver

```
FINO_SOLVER PC_TYPE $1    # either gamg, lu or mumps (read from commandline)
PARAMETRIC c MIN 1 MAX 12 STEP 1
lc = 10/c
FILE msh parametric-%d.msh c
MESH FILE msh
E = 200e3
nu = 0.3
PHYSICAL_GROUP left  BC fixed
PHYSICAL_GROUP right BC Fx=1e4
FINO_STEP
PRINT c lc nodes %e time_wall_total memory time_wall_build time_wall_solve time_wall_stress
```

```
$ for i in gamg lu mumps; do fino parametric-solver.fin $i > `hostname`-${i}.dat; done
$
```



a



b

Figure 5: Wall time needed to solve the linear problem as a function of the number of nodes in two different computers. a — High-end desktop computer, b — Virtual server running on the cloud Fig. 5 shows the dependence of the wall time (`https://en.wikipedia.org/wiki/Elapsed_real_time`) needed to solve the linear problem with respect to the number of nodes in two different computers. Only the time needed to solve the linear problem is plotted. That is to say, the time needed to mesh the geometry, to build the

matrix and to compute the stresses out of the displacements is not taken into account. The reported times correspond to only one process, i.e. Fino is run in serial mode with no parallelization requested. It can be seen that direct solvers are faster than the iterative method for small problems. Yet GAMG scales better and for a certain problem size (which depends on the hardware and more importantly on the particular problem being solved) its performance is better than that of the direct solvers. This is a known result, which can be stated as direct solvers are *robust* but *not scalable*.[1]

Fino defaults to GAMG+GMRES since this combination is provided natively by PETSc and does not need any extra library (as in the MUMPS case), but it still sticks to the rule of optimization (`https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html#rule_of_optimization`). Chances are that another combination of preconditioner, solver (and hardware!) might be better suitable for the problem being solved. It is up to the user to measure and to choose the most convenient configuration to obtain results as efficiently as possible.

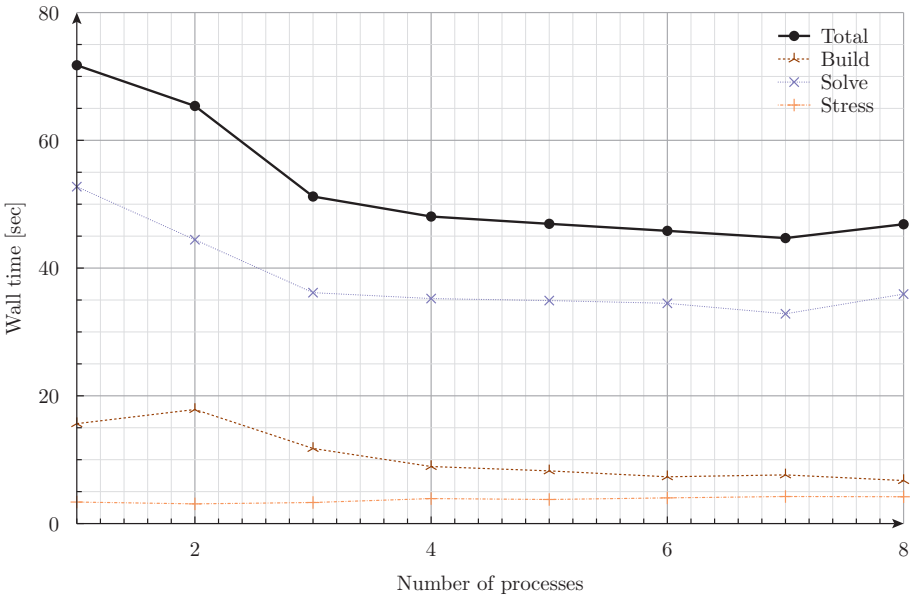


Figure 6: Wall time for strong parallel scaling test.

[1] See second bullet of slide #6 in `http://www.mcs.anl.gov/petsc/petsc-20/tutorial/PETSc3.pdf`.

4 Reference

4.1 Fino keywords

4.1.1 FINO_LINEARIZE

Performs stress linearization according to ASME VII-Sec 5 over a Stress Classification Line

```
FINO_LINEARIZE { PHYSICAL_GROUP <physical_group> | START_POINT <x1> <y1> <z1> END_POINT <x2> <y2> <z2> }
 [ FILE <file_id> | FILE_PATH <file_path> ]
 [ TOTAL { vonmises tresca | tresca | principal1 | principal2 | principal3 }
 [ M <variable> ]
 [ MB <variable> ]
 [ PEAK <variable> ]
```

The Stress Classification Line (SCL) may be given either as a one-dimensional physical group in the mesh or as the (continuous) spatial coordinates of two end-points. If the SCL is given as a `PHYSICAL_GROUP`, the entity should be one-dimensional (i.e a line) independently of the dimension of the problem. If the SCL is given with `START_POINT` and `END_POINT`, the number of coordinates given should match the problem dimension (i.e three coordinates for full 3D problems and two coordinates for axisymmetric or plane problems). Coordinates can be given algebraic expressions that will be evaluated at the time of the linearization. If either a `FILE` or a `FILE_PATH` is given, the total, membrane and membrane plus bending stresses are written as a function of a scalar $t \in [0, 1]$. Moreover, the individual elements of the membrane and bending stress tensors are written within comments (i.e. lines starting with the hash symbol `#`). By default, the linearization uses the Von Mises criterion for the composition of stresses. The definition of what *total stress* means can be changed using the `TOTAL` keyword. The membrane, bending and peak stress tensor elements are combined using the Von Mises criterion and stored as variables. If no name for any of the variables is given, they are stored in `M_group`, `B_group` and `P_group` respectively if there is a physical group. Otherwise `M_1`, `B_1` and `P_1` for the first instruction, `M_2`... etc.

4.1.2 FINO_PROBLEM

Sets the problem type that Fino has to solve.

```
FINO_PROBLEM [ mechanical | thermal | modal ]
 [ AXISYMMETRIC | PLANE_STRESS | PLANE_STRAIN ] [ SYMMETRY_AXIS { x | y } ] [ LINEAR | NON_LINEAR ]
 [ QUASISTATIC | TRANSIENT ]
 [ DIMENSIONS <expr> ] [ MESH <identifier> ]
 [ N_MODES <expr> ]
```

- `mechanical` (or `elastic` or `break`, default) solves the mechanical elastic problem (default).
- `thermal` (or `heat` or `bake`) solves the heat conduction problem.
- `modal` (or `shake`) computes the natural frequencies and oscillation modes.

If the `AXISYMMETRIC` keyword is given, the mesh is expected to be two-dimensional in the x-y plane and the problem is assumed to be axi-symmetric around the axis given by `SYMMETRY_AXIS` (default is y). If the problem type is mechanical and the mesh is two-dimensional on the x-y plane and no axisymmetry is given, either `PLANE_STRESS` and `PLAIN_STRAIN` can be provided (default is plane stress). By default Fino tries to detect wheter the

computation should be linear or non-linear. An explicit mode can be set with either `LINEAR` on `NON_LINEAR`. The number of spatial dimensions of the problem needs to be given either with the keyword `DIMENSIONS` or by defining a `MESH` (with an explicit `DIMENSIONS` keyword) before `FINO_PROBLEM`. If there are more than one `MESH`es define, the one over which the problem is to be solved can be defined by giving the explicit mesh name with `MESH`. By default, the first mesh to be defined in the input file is the one over which the problem is solved. The number of modes to be computed in the modal problem. The default is DEFAULT_NMODES.

4.1.3 FINO_REACTION

Computes the reaction at the selected physical group.

```
FINO_REACTION PHYSICAL_GROUP <physical_group> RESULT { <variable> | <vector> }
```

The result is stored in the variable or vector provided, depending on the number of degrees of freedoms of the problem. If the object passed as `RESULT` does not exist, an appropriate object (scalar variable or vector) is created. For the elastic problem, the components of the total reaction force are stored in the result vector. For the thermal problem, the total power passing through the entity is computed as an scalar.

4.1.4 FINO_SOLVER

Sets options related to the solver and the computation of gradients.

```
FINO_SOLVER [ PROGRESS ]
 [ PC { gamg | mumps | lu | hypre | sor | bjacobi | cholesky | ... } ]
 [ KSP { gmres | mumps | bcgs | bicg | richardson | chebyshev | ... } ]
 [ TS { bdf | arkimex | rosw | glle | beuler | ... } ]
 [ SNES_TYPE { newtonls | newtontr | nrichardson | ngmres | qn | ngs | ... } ]
 [ GRADIENT { gauss | nodes | none } ]
 [ GRADIENT_HIGHER { average | nodes } ]
 [ SMOOTH { always | never | material } ]
 [ ELEMENT_WEIGHT { volume_times_quality | volume | quality | flat } ]
```

If the keyword `PROGRESS` is given, three ASCII lines will show in the terminal the progress of the ensamble of the stiffness matrix (or matrices), the solution of the system of equations and the computation of gradients (stresses). The preconditioner, linear and non-linear solver might be any of those available in PETSc:

- List of `PC`s `http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/PC/PCType.html`.
- List of `KSP`s `http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/KSP/KSPType.html`.
- List of `TS`s `http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/TS/TSType.html`.
- List of `SNES`s `http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/SNES/SNESType.html`.

If either `PC` or `KSP` is set to `mumps` (and PETSc is compiled with MUMPS support) then this direct solver is used. For the mechanical problem, the default is to use GAMG as the preconditioner and PETSc's default solver (GMRES). For the thermal problem, the default is to use the default PETSc settings. For the modal problem, the default is to use the default

SLEPc settings. The `GRADIENT` keyword controls how the derivatives (i.e. strains) at the first-order nodes are to be computed out of the primary unknowns (i.e. displacements).

- `gauss` (default) computes the derivatives at the gauss points and the extrapolates the values to the nodes
- `nodes` computes the derivatives direcetly at the nodes
- `none` does not compute any derivative at all

The way derivatives are computed at high-order nodes (i.e. those at the middle of edges or faces) is controlled with `GRADIENT_HIGHER`:

- `average` (default) assigns the plain average of the first-order nodes that surrond each high-order node
- `none` computes the derivatives at the location of the high-order nodes

The keyword `SMOOTH` controls how the gradient-based functions (i.e. strains, stresses, etc) are smoothed—or not—to obtain nodal values out of data which primarily comes from element-wise evaluations at the Gauss points.

- `always` (default) computes a single value for each node by averaging the contributions of individual elements.
- `never` keeps the contribution of each individual element separate. This option implies that the output mesh is different from the input mesh as each element now has a "copy" of the original shared nodes.
- `material` averages element contribution only for those elements that belong to the same material (i.e. physical group). As with `never`, a new output mesh is created where the nodes are duplicated even for those elements which belong to the same physical group.

The way individual contributions of different elements to the same node are averaged is controlled by `ELEMENT_WEIGHT`:

- `volume_times_quality` (default) weights each element by the product of its volume times its quality
- `volume` weights each element by the its volume
- `quality` weights each element by the its quality
- `flat` performs plain averages (i.e. the same weight for all elements)

4.1.5 FINO_STEP

Ask Fino to solve the problem and advance one step.

```
FINO_STEP
```

The location of the `FINO_STEP` keyword within the input file marks the logical location where the problem is solved and the result functions (displacements, temperatures, stresses, etc.) are available for output or further computation.

4.2 Mesh keywords

4.2.1 MATERIAL

```
MATERIAL <name> [ MESH <name> ] [ PHYSICAL_GROUP <name_1> [ PHYSICAL_GROUP <name_2> [ ... ] ] ] [ <proper
```

4.2.2 MESH

Reads an unstructured mesh from an external file in MSH, VTK or FRD format.

```
MESH [ NAME <name> ] { FILE <file_id> | FILE_PATH <file_path> } [ DIMENSIONS <num_expr> ]
 [ SCALE <expr> ] [ OFFSET <expr_x> <expr_y> <expr_z> ]
 [ INTEGRATION { full | reduced } ] [ RE_READ ]
 [ READ_SCALAR <name_in_mesh> AS <function_name> ] [...]
 [ READ_FUNCTION <function_name> ] [...]
```

If there will be only one mesh in the input file, the `NAME` is optional. Yet it might be needed in cases where there are many meshes and one needs to refer to a particular mesh, such as in `MESH_POST` or `MESH_INTEGRATE`. When solving PDEs (such as in Fino or milonga), the first mesh is the problem mesh. Either a file identifier (defined previously with a `FILE` keyword) or a file path should be given. The format is read from the extension, which should be either

- `.msh` Gmsh ASCII format (`http://gmsh.info/doc/texinfo/gmsh.html#MSH-file-format`), versions 2.2, 4.0 or 4.1
- `.vtk` ASCII legacy VTK (`https://lorensen.github.io/VTKExamples/site/VTKFileFormats/`)
- `.frd` CalculiX's FRD ASCII output (`https://web.mit.edu/calculix_v2.7/CalculiX/cgx_2.7/doc/cgx/node4.html`))

Note than only MSH is suitable for defining PDE domains, as it is the only one that provides information about physical groups. The spatial dimensions should be given with `DIMENSION`. If material properties are uniform and given with variables, the dimensions are not needed and will be read from the file. But if spatial functions are needed (either for properties or read from the mesh file), an explicit value for the mesh dimensions is needed. If either `SCALE` or `OFFSET` are given, the node position if first shifted and then scaled by the provided amounts. For each `READ_SCALAR` keyword, a point-wise defined function of space named `<function_name>` is defined and filled with the scalar data named `<name_in_mesh>` contained in the mesh file. The `READ_FUNCTION` keyword is a shortcut when the scalar name and the to-be-defined function are the same. If no `NAME` is given, the first mesh to be defined is called `first`.

4.2.3 MESH_FILL_VECTOR

Fills the elements of a vector with data evaluated at the nodes or the cells of a mesh.

```
MESH_FILL_VECTOR VECTOR <vector> { FUNCTION <function> | EXPRESSION <expr> }
 [ MESH <name> ] [ NODES | CELLS ]
```

The vector to be filled needs to be already defined and to have the appropriate size, either the number of nodes or cells of the mesh depending on `NODES` or `CELLS` (default is nodes). The elements of the vectors will be either the `FUNCTION` or the `EXPRESSION` of x, y and z evaluated at the nodes or cells of the provided mesh. If there is more than one mesh, the name has to be given.

4.2.4 MESH_FIND_MINMAX

Finds absolute extrema of a function or expression within a mesh-based domain.

```
MESH_FIND_MINMAX { FUNCTION <function> | EXPRESSION <expr> }
 [ MESH <name> ] [ OVER <physical_group_name> ] [ NODES | CELLS ]
```

```
 [ MIN <variable> ] [ MAX <variable> ]
 [ X_MIN <variable> ] [ Y_MIN <variable> ] [ Z_MIN <variable> ] [ I_MIN <variable> ]
 [ X_MAX <variable> ] [ Y_MAX <variable> ] [ Z_MAX <variable> ] [ I_MAX <variable> ]
```

Either a `FUNCTION` or an `EXPRESSION` should be given. In the first case, just the function name is expected (i.e. not its arguments). In the second case, a full algebraic expression including the arguments is expected. If no explicit mesh is provided, the main mesh is used to search for the extrema. If the `OVER` keyword is given, the search is performed only on the provided physical group. Depending on the problem type, it might be needed to switch from `NODES` to `CELLS` but this is usually not needed. If given, the minimum (maximum) value is stored in the variable provided by the `MIN` (`MAX`) keyword. If given, the x (or y or z) coordinate of the minimum (maximum) value is stored in the variable provided by the `X_MIN` (or `Y_MIN` or `Z_MIN`) (`X_MAX`, `Y_MAX`, `Z_MAX`) keyword. If given, the index of the minimum (maximum) value (i.e. the node or cell number) is stored in the variable provided by the `I_MIN` (`I_MAX`) keyword.

4.2.5 MESH_INTEGRATE

Performs a spatial integration of a function or expression over a mesh.

```
MESH_INTEGRATE { FUNCTION <function> | EXPRESSION <expr> }
 [ MESH <mesh_identifier> ] [ OVER <physical_group> ] [ NODES | CELLS ]
 RESULT <variable>
```

The integrand may be either a `FUNCTION` or an `EXPRESSION`. In the first case, just the function name is expected (i.e. not its arguments). In the second case, a full algebraic expression including the arguments is expected. If the expression is just `1` then the volume (or area or length) of the domain is computed. Note that arguments ought to be `x`, `y` and/or `z`. If there are more than one mesh defined, an explicit one has to be given with `MESH`. By default the integration is performed over the highest-dimensional elements of the mesh. If the integration is to be carried out over just a physical group, it has to be given in `OVER`. Either `NODES` or `CELLS` define how the integration is to be performed. In the first case a the integration is performed using the Gauss points and weights associated to each element type. In the second case, the integral is computed as the sum of the product of the function evaluated at the center of each cell (element) and the cell's volume. The scalar result of the integration is stored in the variable given by `RESULT`. If the variable does not exist, it is created.

4.2.6 MESH_MAIN

```
MESH_MAIN [ <name> ]
```

4.2.7 MESH_POST

```
MESH_POST [ MESH <mesh_identifier> ] { FILE <name> | FILE_PATH <file_path> } [ NO_MESH ] [ FORMAT { gmsh
```

4.2.8 PHYSICAL_GROUP

Defines a physical group of elements within a mesh file.

```
PHYSICAL_GROUP <name> [ MESH <name> ] [ DIMENSION <expr> ]
 [ MATERIAL <name> ]
 [ BC <bc_1> <bc_2> ... ]
```

A name is mandatory for each physical group defined within the input file. If there is no physical group with the provided name in the mesh, this instruction makes no effect. If there

are many meshes, an explicit mesh can be given with MESH. Otherwise, the physical group is defined on the main mesh. An explicit dimension of the physical group can be provided with DIMENSION. For volumetric elements, physical groups can be linked to materials using MATERIAL. Note that if a material is created with the same name as a physical group in the mesh, they will be linked automatically. The MATERIAL keyword in PHYSICAL_GROUP is used to link a physical group in a mesh file and a material in the wasora input file with different names. For non-volumetric elements, boundary conditions can be assigned by using the BC keyword. This should be the last keyword of the line, and any token afterwards is treated specially by the underlying solver (i.e. Fino or milonga).

4.2.9 PHYSICAL_PROPERTY

```
PHYSICAL_PROPERTY <name> [ <material_name1> <expr1> [ <material_name2> <expr2> ] ... ]
```

4.3 Special input distributions

TBD.

4.4 Boundary conditions

TBD.

4.5 Result functions

TBD.

4.6 Wasora keywords

4.6.1 =

Assign an expression to a variable, a vector or a matrix.

```
<var>[ [<expr_tmin>, <expr_tmax>] |
<expr_t> ] = <expr> <vector>(<expr_i>)[<expr_i_min, expr_i_max>] [ [<expr_tmin>, <expr_tmax>] |
<expr_t> ] = <expr> <matrix>(<expr_i>,<expr_j>)[<expr_i_min, expr_i_max; expr_j_min, expr_j_max>] [ [<expr_tmin
<expr_t> ] = <expr>
```

4.6.2 _.=

Add an equation to the DAE system to be solved in the phase space spanned by PHASE_SPACE.

```
{ 0[(i[,j]][<imin:imax[;jmin:jmax]>] | <expr1> } .= <expr2>
```

4.6.3 ABORT

Catastrophically abort the execution and quit wasora.

```
ABORT
```

Whenever the instruction ABORT is executed, wasora quits without closing files or unlocking shared memory objects. The objective of this instruction is, as illustrated in the examples, either to debug complex input files and check the values of certain variables or to conditionally abort the execution using IF clauses.

4.6.4 ALIAS

Define a scalar alias of an already-defined indentifier.

```
ALIAS { <new_var_name> IS <existing_object> | <existing_object> AS <new_name> }
```

The existing object can be a variable, a vector element or a matrix element. In the first case, the name of the variable should be given as the existing object. In the second case, to alias the second element of vector `v` to the new name `new`, `v(2)` should be given as the existing object. In the third case, to alias second element (2,3) of matrix `M` to the new name `new`, `M(2,3)` should be given as the existing object.

4.6.5 CALL

Call a previously dynamically-loaded user-provided routine.

```
CALL <name> [ expr_1 expr_2 ... expr_n ]
```

4.6.6 CLOSE

Explicitly close an already-`OPEN`ed file.

```
CLOSE
```

4.6.7 CONST

Mark a scalar variable, vector or matrix as a constant.

```
CONST name_1 [ <name_2> ] ... [ <name_n> ]
```

4.6.8 DEFAULT_ARGUMENT_VALUE

Give a default value for an optional commandline argument.

```
DEFAULT_ARGUMENT_VALUE <constant> <string>
```

If a `$n` construction is found in the input file but the commandline argument was not given, the default behavior is to fail complaining that an extra argument has to be given in the commandline. With this keyword, a default value can be assigned if no argument is given, thus avoiding the failure and making the argument optional.

4.6.9 DIFFERENTIAL

Explicitly mark variables, vectors or matrices as "differential" to compute intial conditions of DAE systems.

```
DIFFERENTIAL { <var_1> <var_2> ... | <vector_1> <vector_2> ... | <matrix_1> <matrix_2> ... }
```

4.6.10 DO_NOT_EVALUATE_AT_PARSE_TIME

Ask wasora not to evaluate assignments at parse time.

```
DO_NOT_EVALUATE_AT_PARSE_TIME
```

4.6.11 FILE

Define a file, either as input or as output, for further usage.

```
< FILE | OUTPUT_FILE | INPUT_FILE > <name> <printf_format> [ expr_1 expr_2 ... expr_n ] [ INPUT | OUTPUT | MODE
```

4.6.12 FIT

Fit a function of one or more arguments to a set of pointwise-defined data.

```
FIT <function_to_be_fitted> TO <function_with_data> VIA <var_1> <var_2> ... <var_n>
 [ GRADIENT <expr_1> <expr_2> ... <expr_n> ]
 [ RANGE_MIN <expr_1> <expr_2> ... <expr_j> ]
 [ RANGE_MAX <expr_1> <expr_2> ... <expr_n> ]
 [ DELTAEPSREL <expr> ] [ DELTAEPSABS <expr> ] [ MAX_ITER <expr> ]
 [ VERBOSE ] [ RERUN | DO_NOT_RERUN ]
```

The function with the data has to be point-wise defined (i.e. a `FUNCTION` read from a file with inline `DATA`). The function to be fitted has to be parametrized with at least one of the variables provided after the `VIA` keyword. Only the names of the functions have to be given, not the arguments. Both functions have to have the same number of arguments. The initial guess of the solution is given by the initial value of the variables listed in the `VIA` keyword. Analytical expressions for the gradient of the function to be fitted with respect to the parameters to be fitted can be optionally given with the `GRADIENT` keyword. If none is provided, the gradient will be computed numerically using finite differences. A range over which the residuals are to be minimized can be given with `RANGE_MIN` and `RANGE_MAX`. The expressions give the range of the arguments of the functions, not of the parameters. For multidimensional fits, the range is an hypercube. If no range is given, all the definition points of the function with the data are used for the fit. Convergence can be controlled by giving the relative and absolute tolreances with `DELTAEPSREL` (default `1e-4`) and `DELTAEPSABS` (default `1e-6`), and with the maximum number of iterations `MAX_ITER` (default 100). If the optional keyword `VERBOSE` is given, some data of the intermediate steps is written in the standard output. The combination of arguments that minimize the function are computed and stored in the variables. So if `f(x,y)` is to be minimized, after a `MINIMIZE f` both `x` and `y` would have the appropriate values. The details of the method used can be found in GSL's documentation (`https://www.gnu.org/software/gsl/doc/html/multimin.html`). Some of them use derivatives and some of them do not. Default method is `gsl_multimin_fminimizer_nmsimplex2`, which does not need derivatives.

4.6.13 FUNCTION

Define a function of one or more variables.

```
FUNCTION <name>(<var_1>[,var2,...,var_n]) { [ = <expr> | FILE_PATH <file_path> | ROUTINE <name> | | MESH <name>
```

The number of variables n is given by the number of arguments given between parenthesis after the function name. The arguments are defined as new variables if they had not been already defined as variables. If the function is given as an algebraic expression, the shorthand operator `:=` can be used. That is to say, `FUNCTION f(x) = x^2` is equivalent to `f(x) := x^2`. If a `FILE_PATH` is given, an ASCII file containing at least $n+1$ columns is expected. By default, the first n columns are the values of the arguments and the last column is the value of the function at those points. The order of the columns can be changed with the keyword `COLUMNS`, which expects $n+1$ expressions corresponding to the column numbers. A function of type `ROUTINE` calls an already-defined user-provided routine using the `CALL` keyword and passes the values of the variables in each required evaluation as a `double *` argument. If `MESH` is given, the definition points are the nodes or the cells of the mesh. The function arguments should be (x), (x, y) or (x, y, z) matching the dimension the mesh. If the keyword `DATA` is used, a new empty vector of the appropriate size is defined. The elements of this new vector can be assigned to the values of the function at the i-th node

or cell. If the keyword `VECTOR` is used, the values of the dependent variable are taken to be the values of the already-existing vector. Note that this vector should have the size of the number of nodes or cells the mesh has, depending on whether `NODES` or `CELLS` is given. If `VECTOR_DATA` is given, a set of $n+1$ vectors of the same size is expected. The first $n+1$ correspond to the arguments and the last one is the function value. Interpolation schemes can be given for either one or multi-dimensional functions with `INTERPOLATION`. Available schemes for $n = 1$ are:

- linear
- polynomial, the grade is equal to the number of data minus one
- spline, cubic (needs at least 3 points)
- spline_periodic
- akima (needs at least 5 points)
- akima_periodic (needs at least 5 points)
- steffen, always-monotonic splines-like (available only with GSL >= 2.0)

Default interpolation scheme for one-dimensional functions is `(*gsl_interp_linear)`.

Available schemes for $n > 1$ are:

- nearest, $f(\vec{x})$ is equal to the value of the closest definition point
- shepard, inverse distance weighted average definition points (`https://en.wikipedia.org/wiki/Inverse_distance_weighting`) (might lead to inefficient evaluation)
- shepard_kd, average of definition points within a kd-tree (`https://en.wikipedia.org/wiki/Inverse_distance_weighting#Modified_Shepard's_method`) (more efficient evaluation provided `SHEPARD_RADIUS` is set to a proper value)
- bilinear, only available if the definition points configure an structured hypercube-like grid. If $n > 3$, `SIZES` should be given.

For $n > 1$, if the euclidean distance between the arguments and the definition points is smaller than `INTERPOLATION_THRESHOLD`, the definition point is returned and no interpolation is performed. Default value is square root of `9.5367431640625e-07`. The initial radius of points to take into account in `shepard_kd` is given by `SHEPARD_RADIUS`. If no points are found, the radius is double until at least one definition point is found. The radius is doubled until at least one point is found. Default is `1.0`. The exponent of the `shepard` method is given by `SHEPARD_EXPONENT`. Default is `2`. When requesting `bilinear` interpolation for $n > 3$, the number of definition points for each argument variable has to be given with `SIZES`, and wether the definition data is sorted with the first argument changing first (`X_INCREASES_FIRST` evaluating to non-zero) or with the last argument changing first (zero). The function can be pointwise-defined inline in the input using `DATA`. This should be the last keyword of the line, followed by $N = k \cdot (n+1)$ expresions giving k definition points: n arguments and the value of the function. Multiline continuation using brackets `{` and `}` can be used for a clean data organization. See the examples.

4.6.14 HISTORY

Record the time history of a variable as a function of time.

```
HISTORY <variable> <function>
```

4.6.15 IF

Begin a conditional block.

```
IF expr
<block_of_instructions_if_expr_is_true>
[ ELSE ]
[block_of_instructions_if_expr_is_false]
ENDIF
```

4.6.16 IMPLICIT

Define whether implicit declaration of variables is allowed or not.

```
IMPLICIT { NONE | ALLOWED }
```

By default, wasora allows variables (but not vectors nor matrices) to be implicitly declared. To avoid introducing errors due to typos, explicit declaration of variables can be forced by giving `IMPLICIT NONE`. Whether implicit declaration is allowed or explicit declaration is required depends on the last `IMPLICIT` keyword given, which by default is `ALLOWED`.

4.6.17 INCLUDE

Include another wasora input file.

```
INCLUDE <file_path> [ FROM <num_expr> ] [ TO <num_expr> ]
```

Includes the input file located in the string `file_path` at the current location. The effect is the same as copying and pasting the contents of the included file at the location of the `INCLUDE` keyword. The path can be relative or absolute. Note, however, that when including files inside `IF` blocks that instructions are conditionally-executed but all definitions (such as function definitions) are processed at parse-time independently from the evaluation of the conditional. The optional `FROM` and `TO` keywords can be used to include only portions of a file.

4.6.18 INITIAL_CONDITIONS_MODE

Define how initial conditions of DAE problems are computed.

```
INITIAL_CONDITIONS_MODE { AS_PROVIDED | FROM_VARIABLES | FROM_DERIVATIVES }
```

In DAE problems, initial conditions may be either:

- equal to the provided expressions (`AS_PROVIDED`)
- the derivatives computed from the provided phase-space variables (`FROM_VARIABLES`)
- the phase-space variables computed from the provided derivatives (`FROM_DERIVATIVES`)

In the first case, it is up to the user to fulfill the DAE system at $t = 0$. If the residuals are not small enough, a convergence error will occur. The `FROM_VARIABLES` option means calling IDA's `IDACalcIC` routine with the parameter `IDA_YA_YDP_INIT`. The `FROM_DERIVATIVES` option means calling IDA's `IDACalcIC` routine with the parameter IDA_Y_INIT. Wasora should be able to automatically detect which variables in phase-space are differential and which are purely algebraic. However, the `DIFFERENTIAL` keyword may be used to explicitly define them. See the (SUNDIALS documentation)[https://computation.llnl.gov/casc/sundials/documentation/ida_guide.pdf] for further information.

4.6.19 LOAD_PLUGIN

Load a wasora plug-in from a dynamic shared object.

```
LOAD_PLUGIN { <file_path> | <plugin_name> }
```

A wasora plugin in the form of a dynamic shared object (i.e. `.so`) can be loaded either with the `LOAD_PLUGIN` keyword or from the command line with the `-p` option. Either a file path or a plugin name can be given. The following locations are tried:

- the current directory `./`
- the parent directory `../`
- the user's `LD_LIBRARY_PATH`
- the cache file `/etc/ld.so.cache`
- the directories `/lib`, `/usr/lib`, `/usr/local/lib`

If a wasora plugin was compiled and installed following the `make install` procedure, the plugin should be loaded by just passing the name to `LOAD_PLUGIN`.

4.6.20 LOAD_ROUTINE

Load one or more routines from a dynamic shared object.

```
LOAD_ROUTINE <file_path> <routine_1> [ <routine_2> ... <routine_n> ]
```

4.6.21 M4

Call the `m4` macro processor with definitions from wasora variables or expressions.

```
M4 { INPUT_FILE <file_id> | FILE_PATH <file_path> } { OUTPUT_FILE <file_id> | OUTPUT_FILE_PATH <file_path> } [
```

4.6.22 MATRIX

Define a matrix.

```
MATRIX <name> ROWS <expr> COLS <expr> [ DATA num_expr_1 num_expr_2 ... num_expr_n ]
```

4.6.23 MINIMIZE

Find the combination of arguments that give a (relative) minimum of a function.

```
MINIMIZE <function>
 [ METHOD { nmsimplex2 | nmsimplex | nmsimplex2rand | conjugate_fr | conjugate_pr | vector_bfgs2 | vector_bfgs
 [ GRADIENT <expr_1> <expr_2> ... <expr_n> ]
 [ GUESS <expr_1> <expr_2> ... <expr_n> ]
 [ MIN <expr_1> <expr_2> ... <expr_n> ]
 [ MAX <expr_1> <expr_2> ... <expr_n> ]
 [ STEP <expr_1> <expr_2> ... <expr_n> ]
 [ MAX_ITER <expr> ] [ TOL <expr> ] [ GRADTOL <expr> ]
 [ VERBOSE ] [ NORERUN ]
```

4.6.24 PARAMETRIC

Systematically sweep a zone of the parameter space, i.e. perform a parametric run.

```
PARAMETRIC <var_1> [ ... <var_n> ] [ TYPE { linear | logarithmic | random | gaussianrandom | sobol | niederreit
```

4.6.25 PHASE_SPACE

Define which variables, vectors and/or matrices belong to the phase space of the DAE system to be solved.

```
PHASE_SPACE { <vars> | <vectors> | <matrices> }
```

4.6.26 PRINT

Print plain-text and/or formatted data to the standard output or into an output file.

```
PRINT [ FILE <file_id> | FILE_PATH <file_path> ] [ NONEWLINE ] [ SEP <string> ] [ NOSEP ] [ HEADER ] [ SKIP_STE
```

Each argument `object` that is not a keyword is expected to be part of the output, can be either a matrix, a vector, an scalar algebraic expression. If the given object cannot be solved into a valid matrix, vector or expression, it is treated as a string literal if `IMPLICIT` is `ALLOWED`, otherwise a parser error is raised. To explicitly interpret an object as a literal string even if it resolves to a valid numerical expression, it should be prefixed with the `TEXT` keyword. Hashes `#` appearing literal in text strings have to be quoted to prevent the parser to treat them as comments within the wasora input file and thus ignoring the rest of the line. Whenever an argument starts with a porcentage sign `%`, it is treated as a C `printf`-compatible format definition and all the objects that follow it are printed using the given format until a new format definition is found. The objects are treated as double-precision floating point numbers, so only floating point formats should be given. The default format is `"%g"`. Matrices, vectors, scalar expressions, format modifiers and string literals can be given in any desired order, and are processed from left to right. Vectors are printed element-by-element in a single row. See `PRINT_VECTOR` to print vectors column-wise. Matrices are printed element-by-element in a single line using row-major ordering if mixed with other objects but in the natural row and column fashion if it is the only given object. If the `FILE` keyword is not provided, default is to write to stdout. If the `NONEWLINE` keyword is not provided, default is to write a newline `\n` character after all the objects are processed. The `SEP` keywords expects a string used to separate printed objects, the default is a tab 'DEFAULT_PRINT_SEPARATOR' character. Use the `NOSEP` keyword to define an empty string as object separator. If the `HEADER` keyword is given, a single line containing the literal text given for each object is printed at the very first time the `PRINT` instruction is processed, starting with a hash `#` character. If the `SKIP_STEP` (`SKIP_STATIC_STEP`)keyword is given, the instruction is processed only every the number of transient (static) steps that results in evaluating the expression, which may not be constant. By default the `PRINT` instruction is evaluated every step. The `SKIP_HEADER_STEP` keyword works similarly for the optional `HEADER` but by default it is only printed once. The `SKIP_TIME` keyword use time advancements to choose how to skip printing and may be useful for non-constant time-step problems.

4.6.27 PRINT_FUNCTION

Print one or more functions as a table of values of dependent and independent variables.

```
PRINT_FUNCTION <function_1> [ { function_2 | expr_1 } ... { function_n | expr_n-1 } ] [ FILE <file_id> | FILE_P
```

4.6.28 PRINT_VECTOR

Print the elements of one or more vectors.

```
PRINT_VECTOR [ FILE <file_id> ] FILE_PATH <file_path> ] [ { VERTICAL | HORIZONTAL } ] [ ELEMS_PER_LINE <expr> ]
```

4.6.29 READ

Read data (variables, vectors o matrices) from files or shared-memory segments.

```
[ READ | WRITE ] [ SHM <name> ] [ { ASCII_FILE_PATH | BINARY_FILE_PATH } <file_path> ] [ { ASCII_FILE | BINARY_
```

4.6.30 SEMAPHORE

Perform either a wait or a post operation on a named shared semaphore.

```
[ SEMAPHORE | SEM ] <name> { WAIT | POST }
```

4.6.31 SHELL

Execute a shell command.

```
SHELL <print_format> [ expr_1 expr_2 ... expr_n ]
```

4.6.32 SOLVE

Solve a non-linear system of n equations with n unknowns.

```
SOLVE <n> UNKNOWNS <var_1> <var_2> ... <var_n> RESIDUALS <expr_1> <expr_2> ... <expr_n> ] GUESS <expr_1> <expr_
```

4.6.33 TIME_PATH

Force transient problems to pass through specific instants of time.

```
TIME_PATH <expr_1> [ <expr_2> [ ... <expr_n> ] ]
```

The time step `dt` will be reduced whenever the distance between the current time `t` and the next expression in the list is greater than `dt` so as to force `t` to coincide with the expressions given. The list of expresssions should evaluate to a sorted list of values.

4.6.34 VAR

Define one or more scalar variables.

```
VAR <name_1> [ <name_2> ] ... [ <name_n> ]
```

4.6.35 VECTOR

Define a vector.

```
VECTOR <name> SIZE <expr> [ DATA <expr_1> <expr_2> ... <expr_n> | FUNCTION_DATA <function> ]
```

4.6.36 VECTOR_SORT

Sort the elements of a vector using a specific numerical order, potentially making the same rearrangement of another vector.

```
VECTOR_SORT <vector> [ ASCENDING_ORDER | DESCENDING_ORDER ] [ <vector> ]
```

4.6.37 WRITE

Write data (variables, vectors o matrices) to files or shared-memory segments. See the `READ` keyword for usage details.

4.7 Fino variables

4.7.1 delta_sigma_max

The uncertainty of the maximum Von Mises stress σ of the elastic problem. Not to be confused with the maximum uncertainty of the Von Mises stress.

4.7.2 displ_max

The module of the maximum displacement of the elastic problem.

4.7.3 displ_max_x

The x coordinate of the maximum displacement of the elastic problem.

4.7.4 displ_max_y

The y coordinate of the maximum displacement of the elastic problem.

4.7.5 displ_max_z

The z coordinate of the maximum displacement of the elastic problem.

4.7.6 fino_abstol

Absolute tolerance of the linear solver, as passed to PETSc's [`KSPSetTolerances`](http: Default `1e-50`.

4.7.7 fino_divtol

Divergence tolerance, as passed to PETSc's [`KSPSetTolerances`](http: Default `1e+4`.

4.7.8 fino_gamg_threshold

Relative threshold to use for dropping edges in aggregation graph for the [Geometric Algebraic Multigrid Preconditioner](http: as passed to PETSc's [`PCGAMGSetThreshold`](http: A value of 0.0 means keep all nonzero entries in the graph; negative means keep even zero entries in the graph. Default `0.01`.

4.7.9 fino_iterations

This variable contains the actual number of iterations used by the solver. It is set after `FINO_STEP`.

4.7.10 fino_max_iterations

Number of maximum iterations before diverging, as passed to PETSc's [`KSPSetTolerances`](http: Default `10000`.

4.7.11 fino_penalty_weight

The weight w used when setting multi-freedom boundary conditions. Higher values mean better precision in the constrain but distort the matrix condition number. Default is `1e8`.

4.7.12 fino_reltol

Relative tolerance of the linear solver, as passed to PETSc's [`KSPSetTolerances`](http: Default `1e-6`.

4.7.13 fino_residual_norm

This variable contains the residual obtained by the solver. It is set after `FINO_STEP`.

4.7.14 lambda

Requested eigenvalue. It is equal to 1.0 until `FINO_STEP` is executed.

4.7.15 memory

Maximum resident set size (global memory used), in bytes.

4.7.16 memory_available

Total available memory, in bytes.

4.7.17 memory_petsc

Maximum resident set size (memory used by PETSc), in bytes.

4.7.18 nodes_rough

The number of nodes of the mesh in `ROUGH` mode.

4.7.19 petsc_flops

Number of floating point operations performed by PETSc/SLEPc.

4.7.20 sigma_max

The maximum von Mises stress σ of the elastic problem.

4.7.21 sigma_max_x

The x coordinate of the maximum von Mises stress σ of the elastic problem.

4.7.22 sigma_max_y

The x coordinate of the maximum von Mises stress σ of the elastic problem.

4.7.23 sigma_max_z

The x coordinate of the maximum von Mises stress σ of the elastic problem.

4.7.24 strain_energy

The strain energy stored in the solid, computed as $1/2 \cdot \vec{u}^T K \vec{u}$ where $\vec{u}$ is the displacements vector and K is the stiffness matrix.

4.7.25 time_cpu_build

CPU time insumed to build the problem matrices, in seconds.

4.7.26 time_cpu_solve

CPU time insumed to solve the problem, in seconds.

4.7.27 time_cpu_stress

CPU time insumed to compute the stresses from the displacements, in seconds.

4.7.28 time_petsc_build

CPU time insumed by PETSc to build the problem matrices, in seconds.

4.7.29 time_petsc_solve

CPU time insumed by PETSc to solve the eigen-problem, in seconds.

4.7.30 time_petsc_stress

CPU time insumed by PETSc to compute the stresses, in seconds.

4.7.31 time_wall_build

Wall time insumed to build the problem matrices, in seconds.

4.7.32 time_wall_solve

Wall time insumed to solve the problem, in seconds.

4.7.33 time_wall_stress

Wall time insumed to compute the stresses, in seconds.

4.7.34 time_wall_total

Wall time insumed to initialize, build and solve, in seconds. CPU time insumed to initialize, build and solve, in seconds. CPU time insumed by PETSc to initialize, build and solve, in seconds.

4.7.35 T_max

The maximum temperature $T_{\max}$ of the thermal problem.

4.7.36 T_min

The minimum temperature $T_{\min}$ of the thermal problem.

4.7.37 u_at_displ_max

The x component u of the maximum displacement of the elastic problem.

4.7.38 u_at_sigma_max

The x component u of the displacement where the maximum von Mises stress σ of the elastic problem is located.

4.7.39 v_at_displ_max

The y component v of the maximum displacement of the elastic problem.

4.7.40 v_at_sigma_max

The y component v of the displacement where the maximum von Mises stress σ of the elastic problem is located.

4.7.41 w_at_displ_max

The z component w of the maximum displacement of the elastic problem.

4.7.42 w_at_sigma_max

The z component w of the displacement where the maximum von Mises stress σ of the elastic problem is located.