Libmesh HOWTO

libmesh-users@lists.sourceforge.net
Revision:

February 2, 2005

1 General usage

1.1 Debug / Profile mode

Debugging and profiler modes can be switched on with `make METHOD=dbg` and `make METHOD=pro`. For syntax checking use `make METHOD=syn`.

1.2 Performance logging

- Create a logger: `PerfLog perf_log ("Matrix Assembly")` Ex4.
- Start logging: `perf_log.start_event("elem init")`
- Stop logging: `perf_log.stop_event("elem init");`

1.3 Petsc-Tools

Use the Petsc tools as command line parameters to the program invocation, e.g. `./myTestProgram -log_summary`. Frequently used options are

- -log_summary: show setup and performance
- -log_info: show setup
- -ksp_monitor: show convergence

2 Basic tasks

2.1 Restart a model

Restarting a model is done by loading a simulation, that has been stored with `equation_systems.write`. It is stored either in a ASCII-File or in the HDF format. It is read with `equation_systems.read`. (Ex2).

2.2 Translate / deform / rotate a mesh

A mesh object can be translated, deformed or rotated with

```
MeshTools::Modification::translate(mesh, 10., 1.);
MeshTools::Modification::rotate(mesh, 90., 10. 0.);
```

2.3 Equation system parameters

Equation system parameters are set with the following methods

es.parameters.set<Real> ("myParam") = 42.;.
es.parameters.set<**unsigned int**> ("linear_solver_maximum_iterations") = 250;

Their values can later be obtained with

Real answer = es.parameters.get<Real> ("myParam");.

2.4 Write to postprocessing file

Libmesh supports many post-processing file types. Writing the mesh is as easy as

GMVIO(mesh).write("out.gmv");

Writing a the mesh together with the current solution is equally simple

GMVIO(mesh).write_equation_systems ("out.gmv", es);

2.5 Add an additional vector, and project it on refined meshes

Add a new vector to a system with `system.add_vector("myvec");`.
Upon mesh refinement, the vector can be projected onto the new mesh with `system.project_vector(system.get_vector`

3 Programming tips

3.1 Autopointer

Automatically take care of a pointer (safely delete it when it goes out of scope) `AutoPtr<FEBase>` (Ex5)

3.2 Scopes

Even in a very simple main program the there need to be scopes so that the variables go out of scope before ending PetSc.

```
libMesh::init (argc, argv);
  {
    Mesh mesh(3);
  }
return libMesh::close();
```