Higher Order Spectral Elements in M3D

Bernhard Hientzsch
Courant Institute of Mathematical Sciences
New York University
mailto:BernHien@gmail.com
http://www.math.nyu.edu/~hientzsc

October 29, 2006

CEMM Meeting Fall 2006
Philadelphia, Pennsylvania

Working with H.R. Strauss at NYU.

Overview

- The plan for M3D and other plans.
- Two examples from the C production code: p-refinement and grid packing for the tilting mode.
- The M3D interface.
- Two examples from M3D: tilting mode and converging a high pressure tokamak equilibrium.
- Status of the M3D integration.
- Some thoughts about (massive) parallelization.

Integration of a spectral element module into M3D

- Proof of concept, algorithmic development in MATLAB. First serial production version in C.
- Refactoring modules and routines and providing two interfaces: opaque pointers for original data structures (SWIG and C) and flat vectors (for FORTRAN interface).
- Reimplement C version, and implement PYTHON and FORTRAN versions using these modules and interfaces (easy scripting).
- Define interface to M3D for discretization modules.
- Implement this interface for spectral element discretizations based on the previous C version.
- Integrate, test, and use this module with serial, OpenMP, PETSc and/or other distributed memory version of M3D.

Other recent work and plans

- More explorative runs and tests with the C version.
- Integrating adaptivity (changing order and changing element grid)?
- Iterative methods and domain decomposition preconditioners for Poisson-type and Helmholtz-type problems. [Have some tests, algorithms, and ideas. No real code yet.]
- Extending solvers to Neumann boundary conditions and other problems possibly needed in future or other M3D versions and/or applications [Have some algorithms, but not in the production code yet.]
- Thinking about distributed memory parallelization of the current direct sparse solver based approach and massively parallel implementations. [Have some ideas. Multi-level Schur?]

Possible further plans

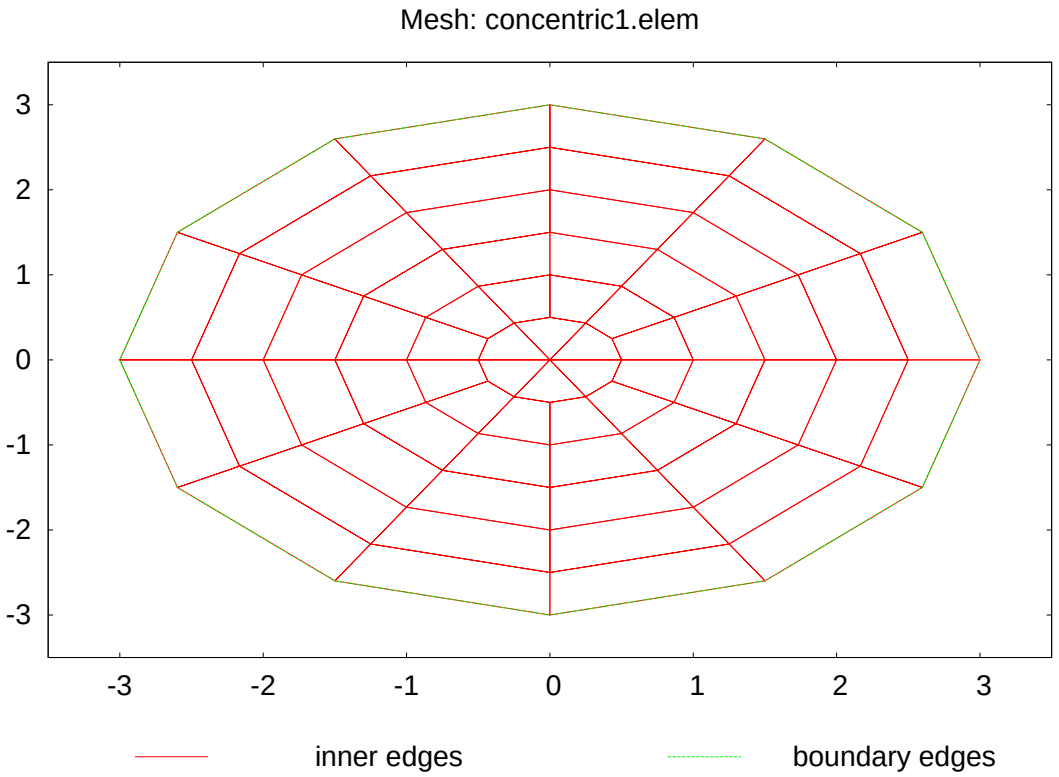
- *C^1-continuous elements*: Implementation and tests in C. [Have some initial MATLAB implementation for a variant.]
- *Domain decomposition preconditioners*. [Have notes and some tests.]
- *Higher order time-stepping* and/or *semi-implicit* schemes. [Have references and some notes.]
- *Fully implicit time-stepping*. [Have some code, still close to proof-of-concept.]
- *Other test problems*. [Have some notes, no full implementations.]
- *Larger and other PDE systems*. [Have some implementation of 5 equation models in C production code/PYTHON scripts driving SWIGified C module.]

Algorithm for C code

- Semi-implicit, leap-frog type time-stepping.
- Only Helmholtz, Laplace equations to be solved.
- Mass matrix, stiffness matrix, and Poisson bracket must be implemented.
- Direct solver. Cholesky, taking advantage of the (block) sparse symmetric structure of global Schur complement for global solves, and of dense Cholesky solvers for the dense local problems. Static condensation.

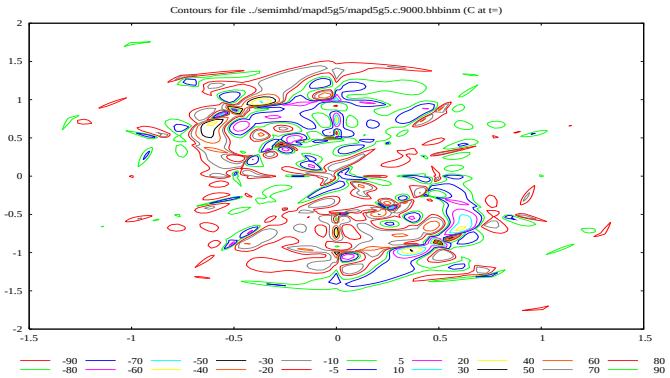
Gaining resolution by increasing degree (p-refinement)

Tilting mode, RMHD (2D) with:
$\Delta t = 0.001$, $\epsilon = 0.001$, $\mu = 0.005$, $\eta = 0.001$.

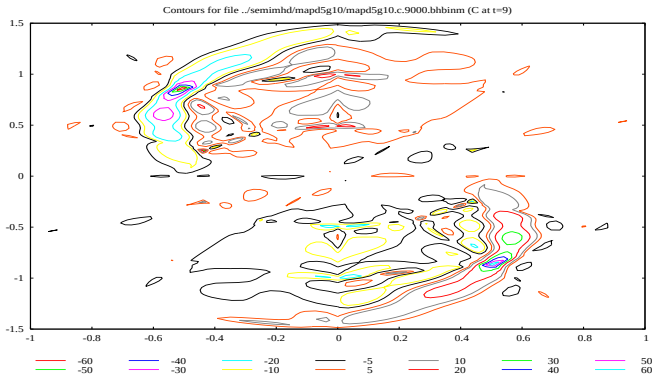


Increasing degree, $t = 9.0$

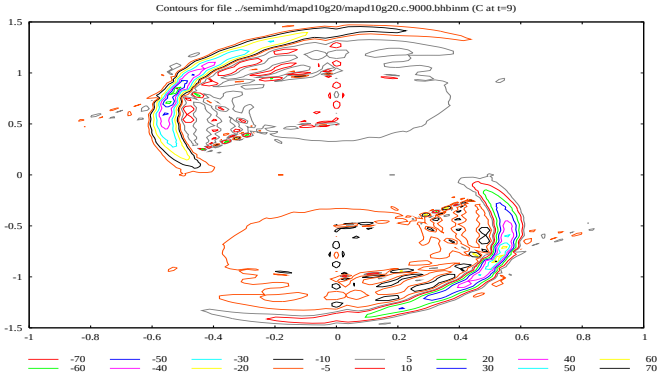
deg=5 intdeg=5



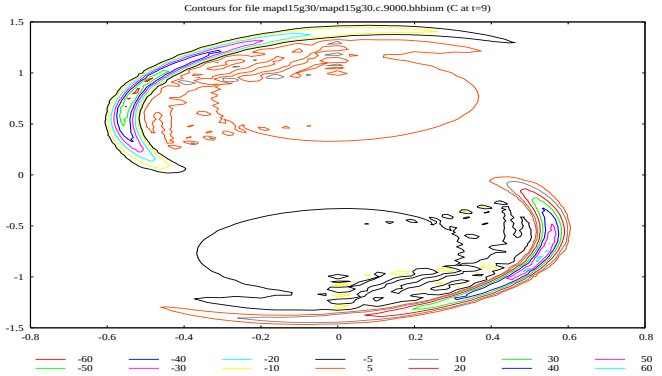
deg=5 intdeg=10



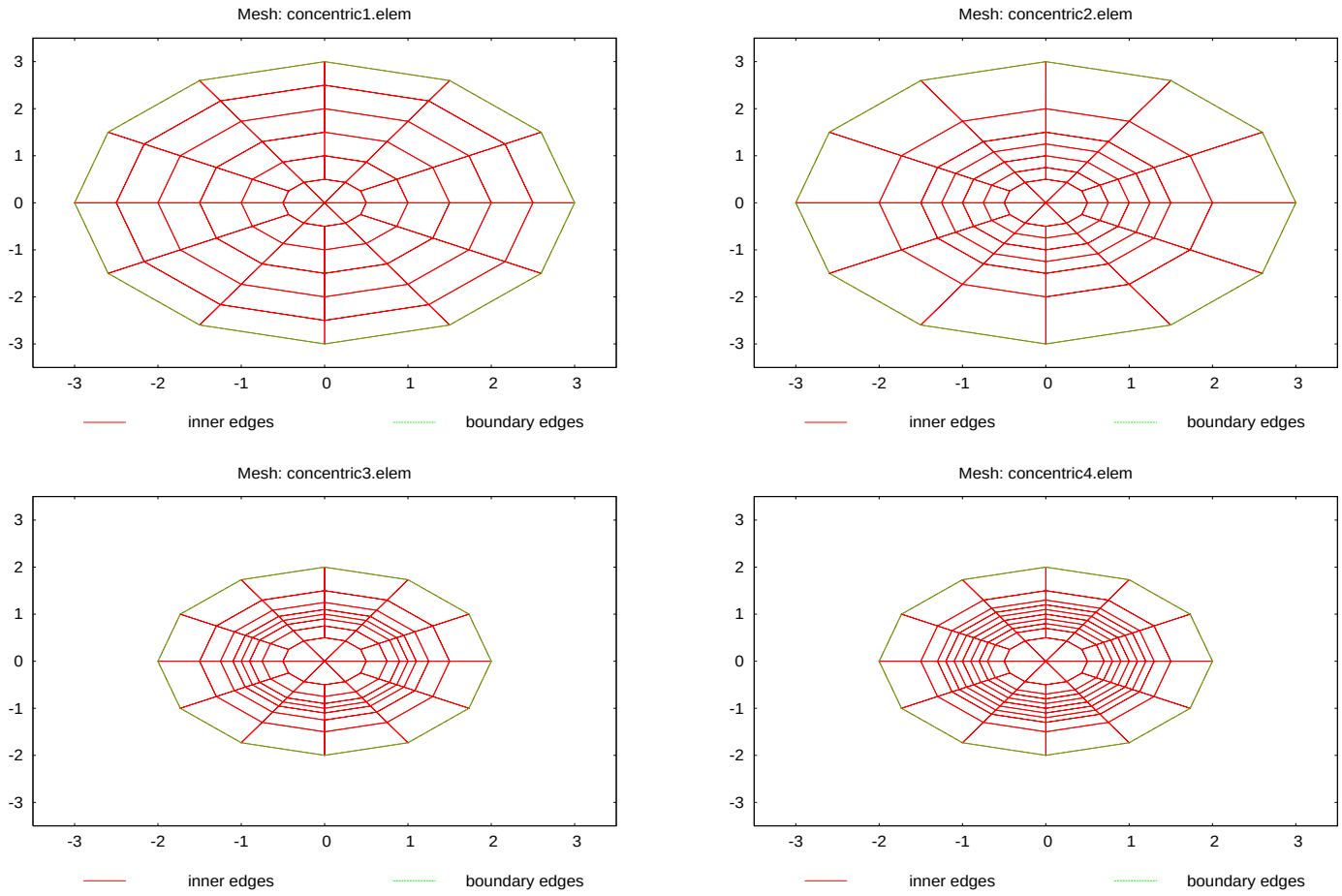
deg=10 intdeg=20



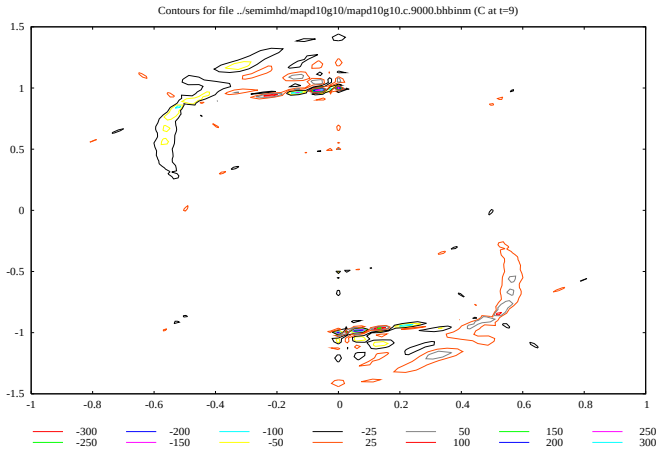
deg=15 intdeg=30

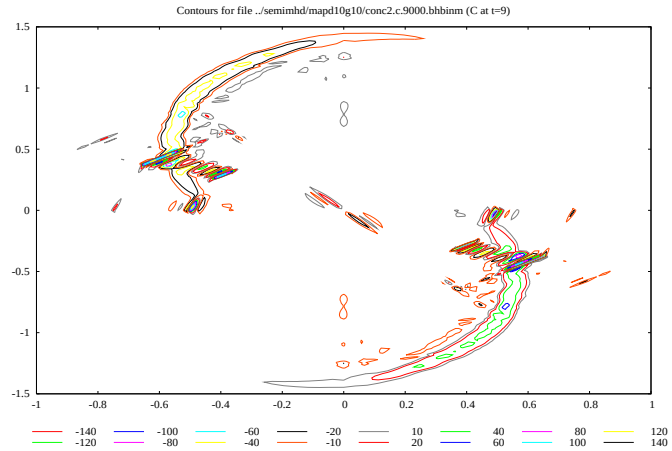


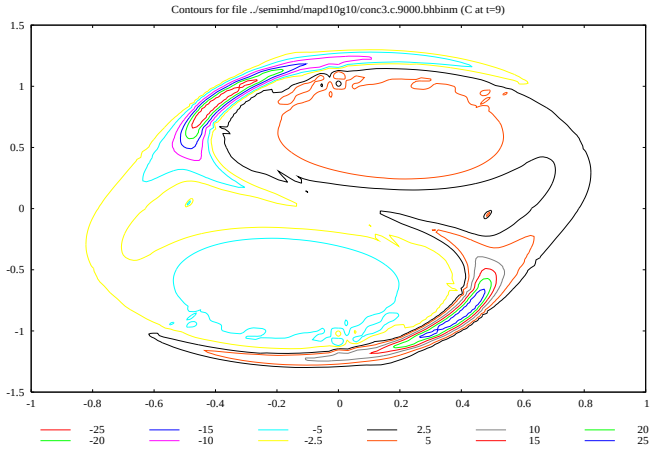
Gaining resolution by packing/adapting the grid

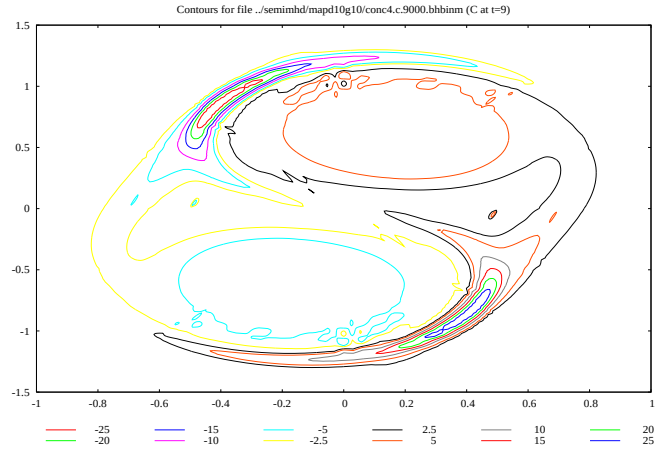


Packing the grid: Current at $t = 9.0$

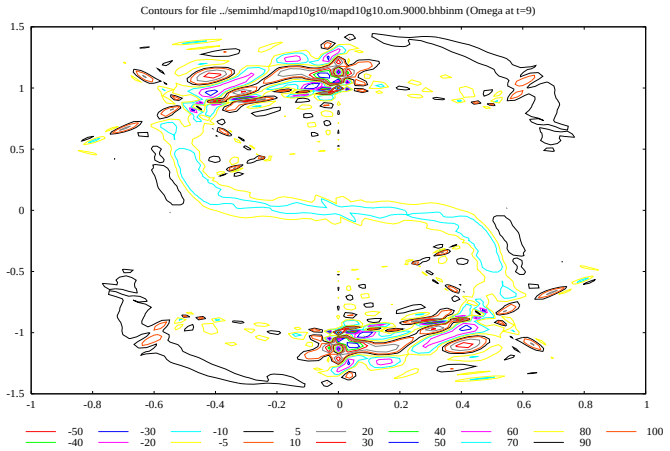


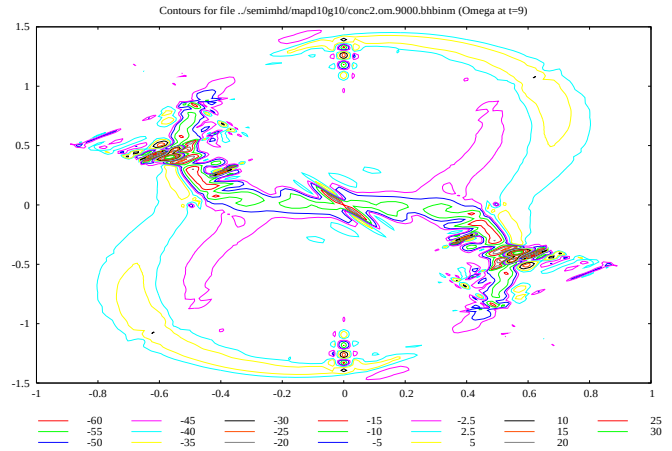


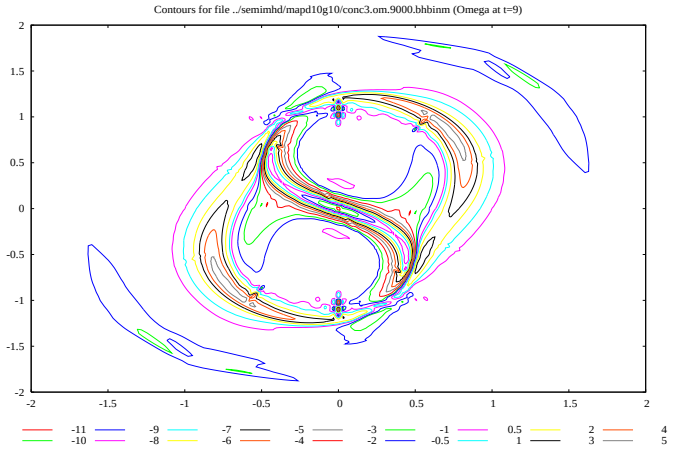


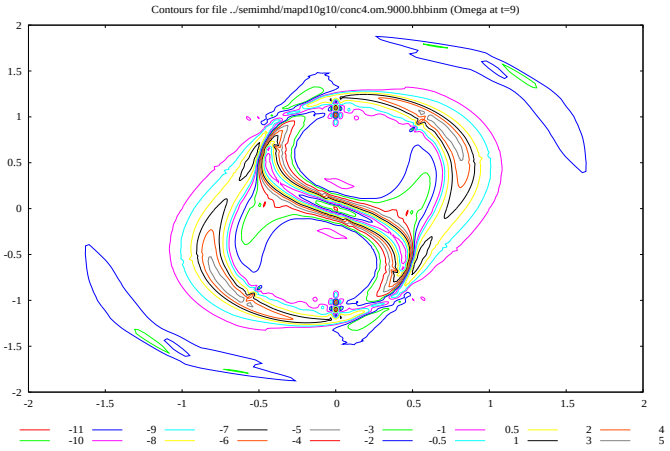


Packing the grid: Ω at $t = 9.0$









M3D interface

- M3D implements the assembly of the right hand side and the elliptic solves in terms of a number of functions/routines defined originally in `mpp3.F`.
- The M3DSEL module provides an implementation of that interface in terms of solvers for specific symmetric statically condensed systems (in a new file `mpp4bh.F`) and spectral element operators.
- The functions in `mpp4bh.F` are implemented in terms of the functions of the original C version (plus some necessary extensions) in `mpp4interface.c`.
- To initialize the discretization and mesh structures and some structures that will be needed in the solvers, we implement a call back function `getselmesh` in getmesh.c which initializes the needed structures, computes the GLL mesh, and sets up the environment for the C module.

The mpp3/mpp4bh interface

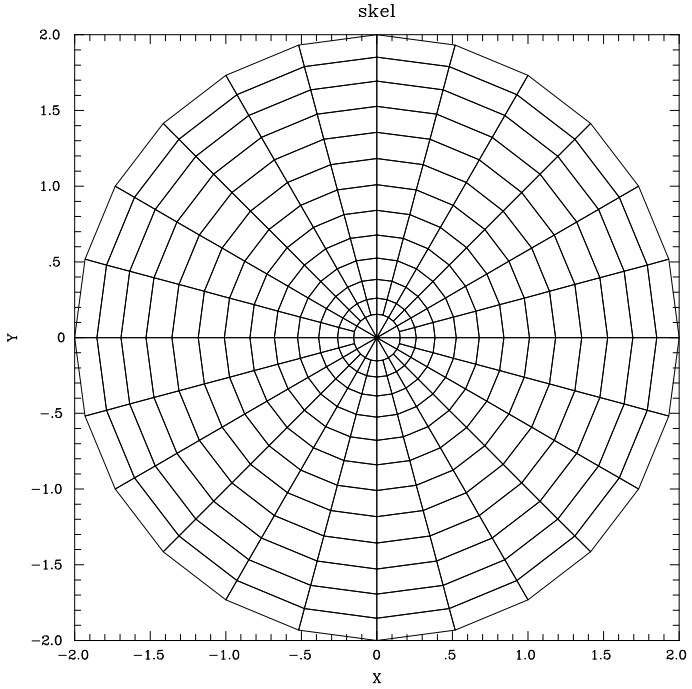
`poiss`	Solves $\Delta u = f$
`delsq`	Computes Δu
`gcro`	Computes $[u, v]$
`gradsq`	Computes $\nabla u \cdot \nabla v$
`agrad`	Computes $\nabla u \cdot \vec{v}$
`dxdr` , `dxdz`, `wgrad`	Compute derivatives
`lowpois`	Solves $\frac{1}{R}(\nabla \cdot (R\nabla))u = f$
`lowpoisa`	Solves $R(\nabla \cdot (\frac{1}{R}\nabla))u = f$
`poisvmu`	Solves $(\Delta - 1/(dt * ss(.)))u = f$
`poisdmd`	Solves $(\nabla \cdot (ss(.)\nabla) - 1/(dt * hmt))u = f$
`poisvmu3`	Solves $(\Delta - 1/(dt * ss(.))u = f$
`lopoismu`	Solves $(\nabla \cdot (\frac{1}{R}\nabla) - 1/(dtt * R * ss(.)))u = f$
`load3`	Computes load vector
`pvol`	Computes volume integral
`intgrsq`	Computes square integral
`*fft*`	Some FFT functions

The mpp4interface.c implementation

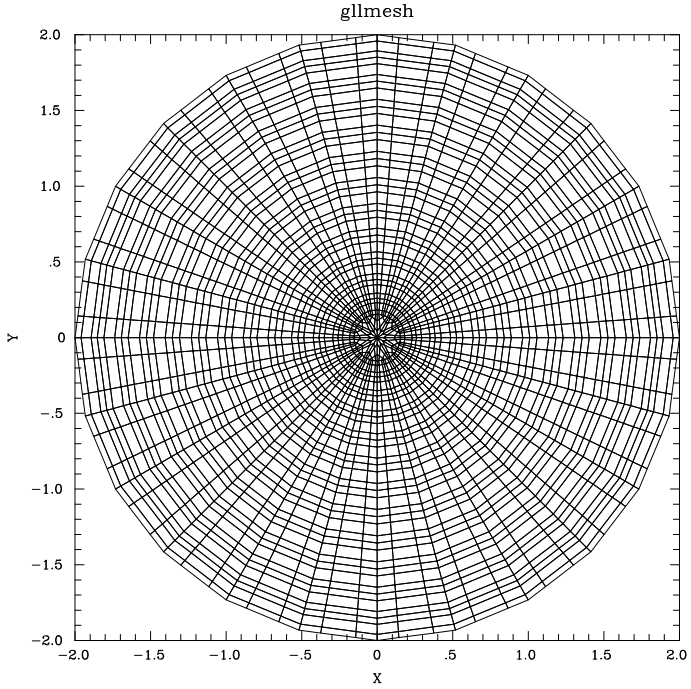
- Differential and integral operators are straightforward to implement once some basic structures have been initialized and some fields have been computed.
- The elliptic solvers are all symmetric and positive definite (or negative definite).
- Assembling local matrices straightforward. Schur complement matrix can be assembled in general function with a function argument (that function computes parametrized local matrix).
- Solvers can be initialized and solve can be implemented as general Schur complement solvers for symmetric systems, using sparse direct solvers for global block-sparse system, dense direct solvers for local systems, and fast BLAS. Use abstract interface for these linear algebra algorithms. Has been implemented with several different packages.

Tilting mode in M3D: the meshes

Element (skeleton) mesh

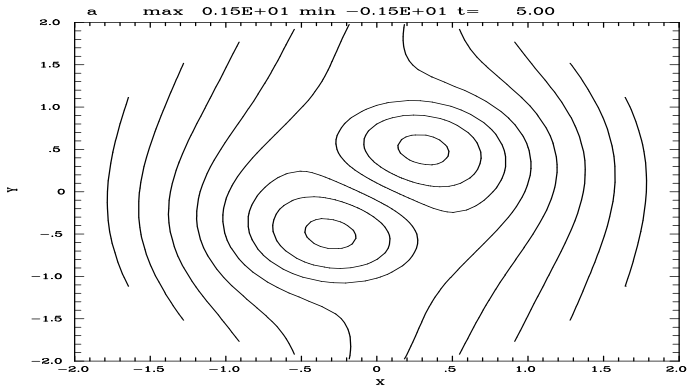


GLL mesh for degree 5

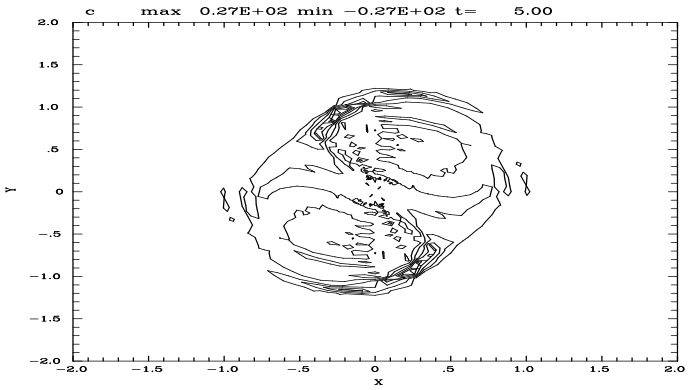


Tilting mode in M3D: results at $t = 5.0$

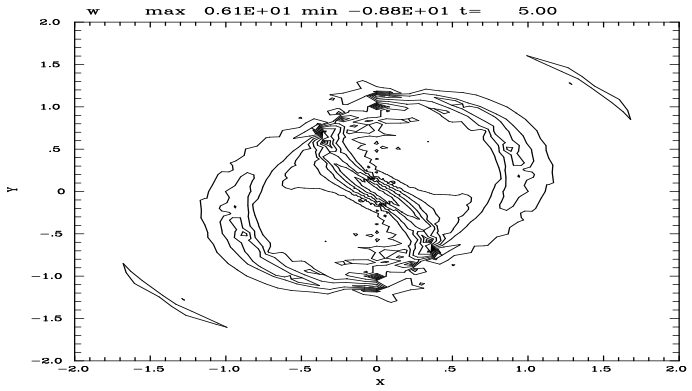
Ψ



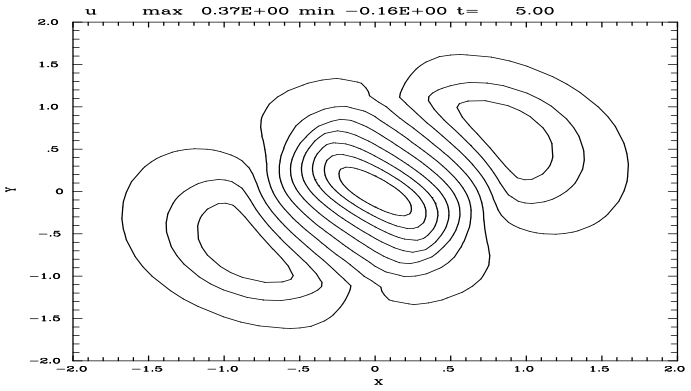
C



Ω

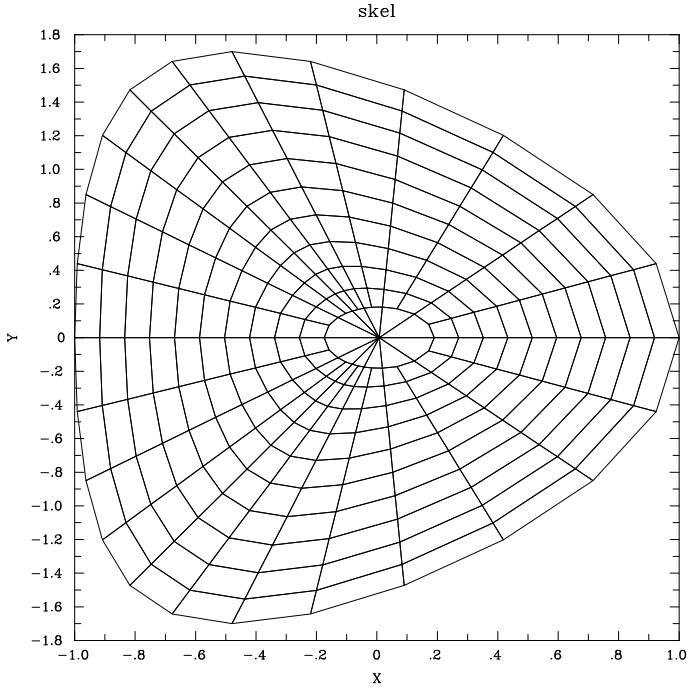


Φ

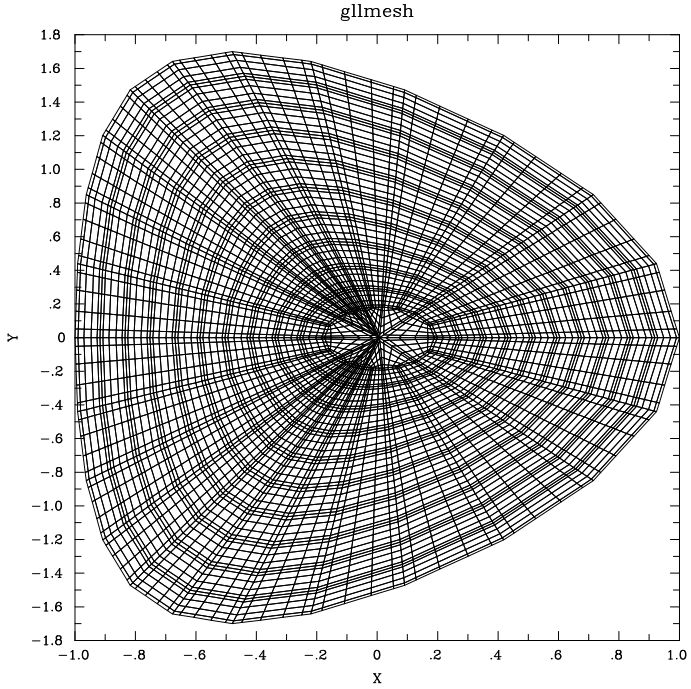


High pressure tokamak equilibrium: the meshes

Element (skeleton) mesh

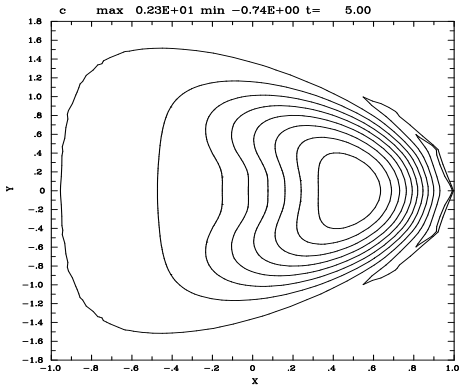


GLL mesh for degree 5

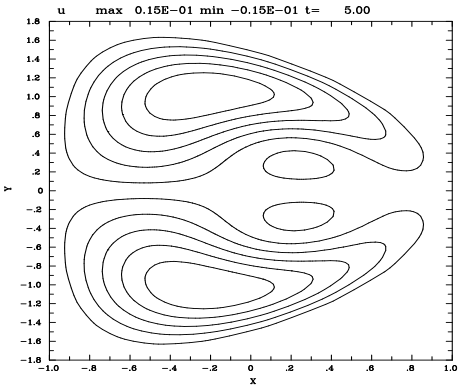


High pressure tokamak equilibrium: results at $t = 5.0$

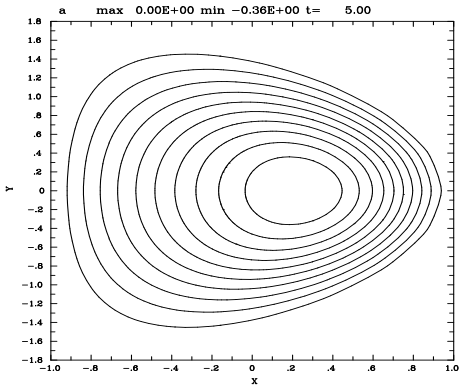
C



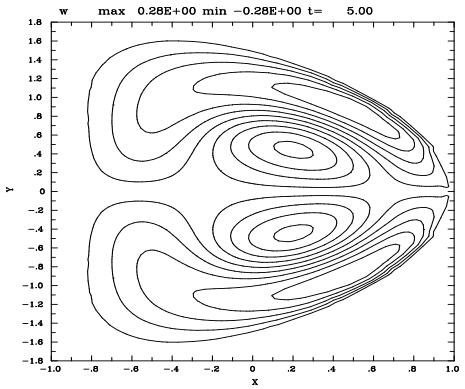
Φ



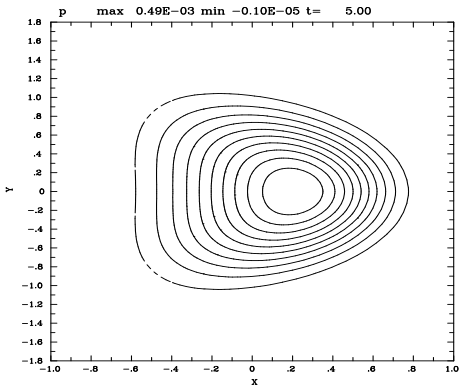
Ψ



Ω

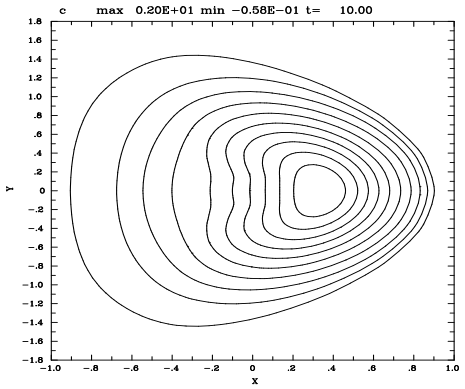


p

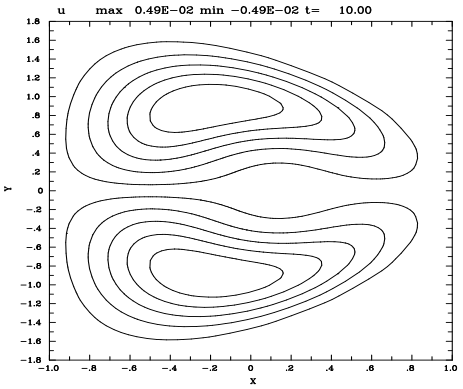


High pressure tokamak equilibrium: $t = 10.0$

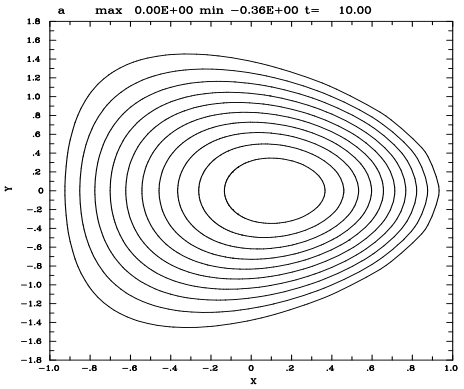
C



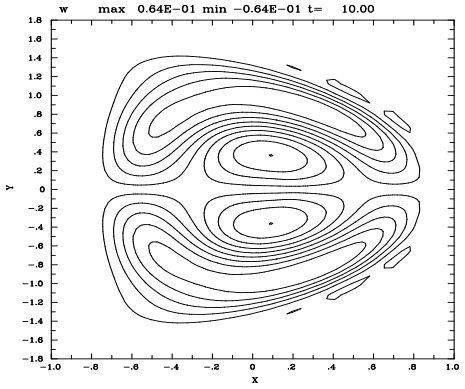
Φ



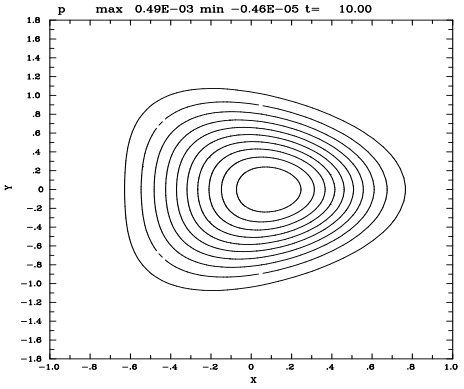
Ψ



Ω

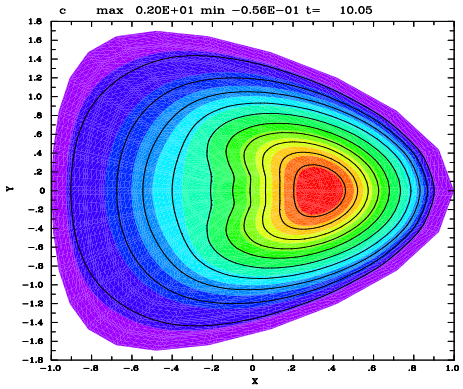


p

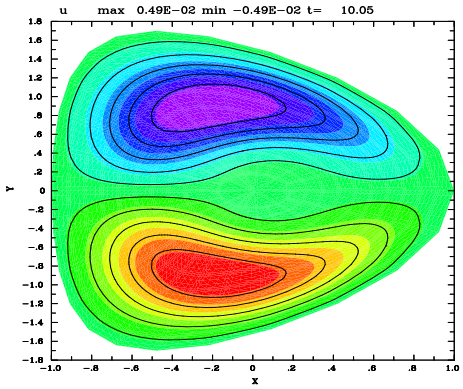


High pressure tokamak equilibrium: $t = 10.05$

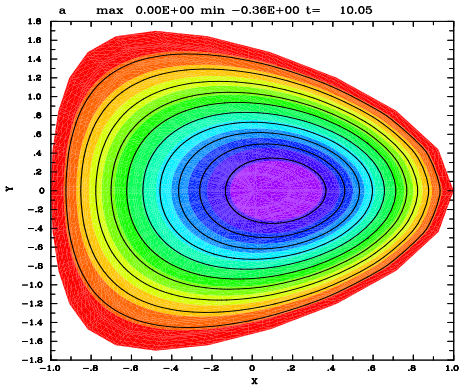
C



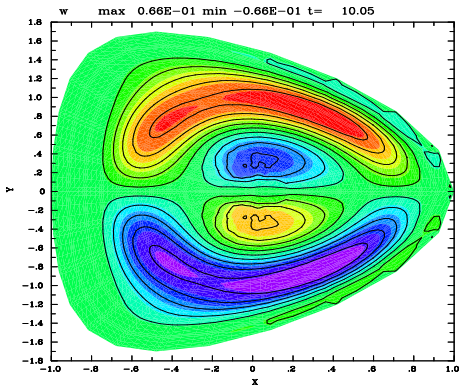
Φ



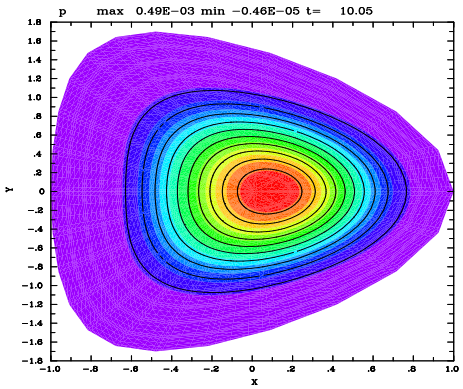
Ψ



Ω



p



Status of the M3D integration: serial, OpenMP

- Serial M3D runs for several test problems. Could be more stable and cleaner, but it works with M3D if done/used right. (As seen in the pictures.)
- In the moment, we are testing and debugging the OpenMP version. Some problems with non-thread-safe/non-shared-memory-safe code in the module. Straightforward duplication of data seems to be the fix.
- Still some work to do to get a nice version which could be disseminated and run without too much hand-holding, but we are currently making quite some progress.

... onto distributed memory, massively parallel

- In the moment, I am considering several possible distributed memory parallelizations of the module and M3D.
- The approach with the least amount of rewriting necessary for the module would be to implement a parallelization of that Schur complement approach, maybe two-level or multi-level, and see if this allows massively parallel implementations. In this case, one only needs to communicate the data on the interfaces, which can be implemented in a straightforward way.
- The approach more similar to PETSc-M3D and possibly optimal with respect to complexity would be to use iterative solvers and optimal domain decomposition/multigrid preconditioners. Alas, Schur complement methods for spectral elements require good coarse spaces but (relatively little) communication since only on the interface; overlapping methods are somewhat easier but require more communication.