Progress on Spectral Elements for 2D MHD

Bernhard Hientzsch
Courant Institute of Mathematical Sciences
New York University
mailto:Bernhard.Hientzsch@na-net.ornl.gov
http://www.math.nyu.edu/~hientzsc

October 22, 2005

CEMM Meeting October 2005
47th Annual Meeting of the Division of Plasma Physics
American Physical Society
October 22-28, 2005 in Denver, Colorado

Working with H.R. Strauss at NYU.

Progress report

- *Rectangular mesh of elements*: C in production mode, ATLAS/LAPACK: fast, workhorse. MATLAB version complete. Different choices for analytical formulae for brackets, general integration of the bracket terms, filtering. Diagnostics. Lots of data: will show on laptop some pictures for different resolutions, some movies.

- *Mapped elements* (straightline, isoparametric). MATLAB: runs. Preproccessing: Deriving mapping/elements from description. (Need to look at M3D.) Rather slow. Laplace and Helmholtz: exponential convergence. C: active development. Static condensation. SuperLU for the large linear system. Some parts run (no complete results).

- C^1*-continuous elements*: on rectangular mesh, same algebraic structure, same solvers. MATLAB version runs (no complete results, not on laptop). C: parts run, debugging.

Why spectral elements: the approach

- Exponential convergence for (standard) problems with (piecewise) smooth solutions. Alignment, postprocessing, filtering etc. for other problems and solutions might restore exponential convergence.
- Fast application of stiffness and mass matrix. Fast solvers for rectangular elements and rectangles. Helmholtz: generalized Sylvester equation.
- Relatively straightforward implementation. Expressed in matrix-matrix multiplications, element-wise multiplications and a few other matrix and vector operations. Translates into matrix-heavy MATLAB or C/C++/FORTRAN using LAPACK/BLAS/...
- Can run at high percentage of peak at modern computer architectures. (Sparse block matrix with small dense blocks.)

Incompressible resistive MHD: primitive variables

∇u is gradient $\nabla u := \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right)$

$\nabla^2 u$ is Laplacian $\nabla^2 u := \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$

$\mathbf{B}$: magnetic field.$\mathbf{v}$: velocity field.
ρ: density, here assumed constant. μ: viscosity. η: resistivity.

$$\frac{\partial \mathbf{B}}{\partial t} = \mathbf{curl}(\mathbf{v} \times \mathbf{B}) + \eta \nabla^2 \mathbf{B} \quad (1)$$

$$\rho \frac{\partial \mathbf{v}}{\partial t} = -\rho \mathbf{v} \cdot \nabla \mathbf{v} + \mathbf{curl}\, \mathbf{B} \times \mathbf{B} + \rho \mu \nabla^2 \mathbf{v} \quad (2)$$

$$\nabla \cdot \mathbf{v} = 0 \quad (3)$$

$$\nabla \cdot \mathbf{B} = 0 \quad (4)$$

Incompressible MHD: potential form in 2D (vorticity-flux)

$[a,b] := \frac{\partial a}{\partial x}\frac{\partial b}{\partial y} - \frac{\partial b}{\partial x}\frac{\partial a}{\partial y}$. Poisson bracket.

$\mathbf{v} = \mathbf{curl}\,\phi$ with velocity flux ϕ. $\mathbf{B} = \mathbf{curl}\,\psi$ with magnetic flux ψ.

Ω: vorticity. C: current density.

$$\frac{\partial \Omega}{\partial t} = [C,\psi] - [\Omega,\phi] + \mu\nabla^2\Omega \quad (5)$$

$$\frac{\partial \psi}{\partial t} = -[\psi,\phi] + \eta\nabla^2\psi \quad (6)$$

$$\nabla^2\phi = \Omega \quad (7)$$

$$C = \nabla^2\psi \quad (8)$$

Incompressible MHD: potential form in 2D (vorticity-current)

$$\frac{\partial \Omega}{\partial t} = [C, \psi] - [\Omega, \phi] + \mu \nabla^2 \Omega \tag{9}$$

$$\frac{\partial C}{\partial t} = [\phi, C] + 2\left[\frac{\partial \phi}{\partial x}, \frac{\partial \psi}{\partial x}\right] + 2\left[\frac{\partial \phi}{\partial y}, \frac{\partial \psi}{\partial y}\right] + [\Omega, \psi] + \eta \nabla^2 C \tag{10}$$

$$\nabla^2 \phi = \Omega \tag{11}$$

$$\nabla^2 \psi = C \tag{12}$$

Time discretization (vorticity-flux, semi-implicit)

Diffusive terms implicit, all others explicit. Leapfrog, i.e., use most current variables. Vorticity-current is discretized in the same way, just more brackets.

$$\frac{\Omega^{n+1}-\Omega^n}{\Delta t} = [C^n,\psi^n]-[\Omega^n,\phi^n]+\mu\nabla^2\Omega^{n+1} \quad (13)$$

$$\nabla^2\phi^{n+1} = \Omega^{n+1} \quad (14)$$

$$\frac{\psi^{n+1}-\psi^n}{\Delta t} = -[\psi^n,\phi^{n+1}]+\eta\nabla^2\psi^{n+1} \quad (15)$$

$$C^{n+1} = \nabla^2\psi^{n+1} \quad (16)$$

Time-stepping code fragment

```
omexrhs=pbrh(c,psi,disc)+pbrh(phi,omega,disc)+soOm;
omrhs=omega+dt*omexrhs;
omega=SolveHH(omrhs,disc,hhdisco);
phi=SolveSLap(omega,disc);
psrhs=dt*(pbrh(phi,psi,disc)+soPs-eta*ApplyLap0Bc(psi,disc));
dpsi=SolveHH(psrhs,disc,hhdiscp);
psi=psi+dpsi;
c=ApplySLap0Bc(psi,disc);
```

Intermediate results are put into variables because of other algorithmic possibilities not shown here. `pbrh` is function handle (MATLAB) or linked against different implementations (C). `disc` is a context structure for the discretization containing one-dimensional matrices etc. `hhdisco`/`hhdiscp` are context structures for the fast Helmholtz solves. All solves: zero Dirichlet data. BCs from tilting mode problem.

Spectral elements, Introduction

- Approximate function in elements by high-order polynomials from tensor product space. Parametrize by values on (mapped) GLL grid.
- Differentiation, interpolation between grids, exact and approximate integrals of products $\mapsto$ matrices acting on values on grid.
- On multi-dimensional non-distorted rectangular elements, matrices are tensor products matrices. F.i., x-derivative in 2D in element aligned with axes is $(D \otimes I)$: derivative only acts along one coordinate.
- Bilinearly or isoparametrically mapped elements: operators are products of block tensor product matrices and diagonal matrices.
- $(A \otimes B)u$ can be rewritten as matrix-matrix multiplication AUB^T, where U and result are 2D arrays instead of vectors.

Spectral elements, Helmholtz

- PDE: $u + \alpha\nabla^2 u = f$, integrate by parts to obtain weak form: $\forall v \in H_0^1$ find $u \in H_0^1$ satisfying $(u, v) - \alpha(\nabla u, \nabla v) = (u, v) - \alpha(u_x, v_x) - \alpha(u_y, v_y) = (f, v)$.
- Instead of H_0^1, use polynomial space and matrices: $v^T M u - \alpha v^T D_x^T M D_x u - \alpha v^T D_y^T M D_y u = v^T M f$.
- $(M^x \otimes M^y)u - \alpha(D^{x,T} \otimes I)(M^x \otimes M^y)(D^x \otimes I)u - \alpha(I \otimes D^{y,T})(M^x \otimes M^y)(I \otimes D^y)u = (M^x \otimes M^y)f$
- $(M^x \otimes M^y - \alpha K^x \otimes M^y - \alpha M^x \otimes K^y)u = (M^x \otimes M^y)f$
- $((\frac{1}{2}M^x - \alpha K^x) \otimes M^y + M^x \otimes (\frac{1}{2}M^y - \alpha K^y))u = (M^x \otimes M^y)f$
- $(A^x \otimes M^y + M^x \otimes A^y)u = (M^x \otimes M^y)f$ or $A^x U M^{y,T} + M^x U A^{y,t} = M^x F M^{y,T}$ (Generalized Sylvester equation.)

Poisson brackets: different analytical expressions

Same continuous object/operator if functions are smooth enough, but different expression. These different expressions lead to different discretizations with different properties.

$$[\phi,\psi] := \phi_x\psi_y - \phi_y\psi_x$$
$$\mathsf{div}(\psi\hat{z}\times\nabla\phi) = \mathsf{div}(-\psi\phi_y,\psi\phi_x,0) = [\phi,\psi] = -[\psi,\phi]$$

$$\begin{aligned}
[\phi,\psi] &= \mathsf{div}(\psi\hat{z}\times\nabla\phi) = -\,\mathsf{div}(\psi\,\mathbf{curl}\,\phi) = -\,\mathsf{div}(\phi\,\mathbf{curl}\,\psi)\\
&= \mathsf{div}(\psi\hat{z}\times\nabla\phi) = -\,\mathsf{div}(\phi\hat{z}\times\nabla\psi)\\
&= \mathbf{curl}(\phi\hat{z})\cdot\nabla\psi = \mathbf{curl}(\psi\hat{z})\cdot\nabla\phi
\end{aligned}$$

Poisson brackets: discretization I

Need to approximate $([a,b],c) = \left(\frac{\partial a}{\partial x}\frac{\partial b}{\partial y} - \frac{\partial a}{\partial y}\frac{\partial b}{\partial x}, v\right)$

In general $[a,b]$ is a nonlinear term, since both a and b are usually dependent variables in the system. $[a,b]$ could be computed exactly as a polynomial, but with much higher degree, if the nonlinearity is only of finite order. Afterwards, that needs to be projected/filtered back.

First choose the analytical expression to start with. Possible choices are:

- Definition of Poisson bracket: $\frac{\partial a}{\partial x}\frac{\partial b}{\partial y} - \frac{\partial a}{\partial y}\frac{\partial b}{\partial x}$
- Use divergence form and integrate by parts (Glasser), for instance $[a,b],c = (\mathrm{div}(b\hat{z} \times \mathbf{grad}\, a), c) = -(b\hat{z} \times \nabla a, \nabla c) +$ boundary terms
- curl-grad form - possible alternative, but don't know of good reason to use it.

Using Poisson brackets: Implementation

Procedure:

- Given a and b, approximate $[a, b]$ on some grid/polynomial space. Can filter it there.
- Integrate by appropriate quadrature rule (after interpolating $[a, b]$ to it with appropriate spectral element interpolation). This also maps $[a, b]$ to its action on a lower-degree space, and therefore truncates/projects $[a, b]$ to a lower-order approximation.
- Filter the Poisson bracket, or
- Filter the entire right hand side at once.

Using Poisson brackets: GLL implementation

If the different representations are all GLL representations, $[a, b]$ is approximated appropriately, and integration happens on the GLL grids, the discretization of $([a, b], c)$ can be written in terms of matrices and operations

$$([a,b],v) \approx v^T I_n^{n_1,T} M^{n_1} F_{n_1}^{n_2} \left((D_x I_n^{n_2} a) \circledast (D_y I_n^{n_2} b) - (D_y I_2^{n_2} a) \circledast (D_x I_n^{n_2} b) \right)$$

Experimenting with different filters such as L^2 projections, an interpolatory filter proposed by Paul Fischer, spectral filtering and diagnostic (transforming to spectral coefficient space for the modal expansion) and an L^2 projection into a C^1 continuous global space.

This is ongoing work, looking at the different polution effects and numerical instabilities that occured in my computer experiments and those of others.

Tilting mode problem - Setup

Introduce polar coordinates $x = r\cos\theta$, $y = r\sin\theta$ and use separable form in polar coordinates,

$$\psi(t=0) = \psi_{0,rad}(r)\cos\theta \qquad \Omega(t=0) = \epsilon\Omega_{0,pert}(r)$$

with the radial functions (k being the first positive zero of J_1, $k \approx 3.8317$)

$$\psi_{0,rad}(r) = \begin{cases} \frac{2J_1(kr)}{kJ_0(k)} & \text{for } r \leq 1 \\ \frac{r^2-1}{r} & \text{for } r \geq 1 \end{cases} \qquad \Omega_{0,pert}(r) = 4(r^2-1)\exp(-r^2)$$

The following boundary conditions were used in this problem:

$$\phi = 0 \qquad C = 0 \qquad \frac{\partial\psi}{\partial t} = 0 \qquad \Omega = 0$$

Some pictures and movies

Switching to other programs to show results ...

C^1 continuous elements

Idea: in early runs, most edge effects and pollution are concentrated close to the boundary and seem to be associated to jumps in the derivative, which are allowed for discretizations with C^0 finite elements.

Such jumps cannot occur if we enforce C^1 continuity. The problem, however, might just manifest itself differently in such an approach. Therefore we designed and implemented a spectral element with C^1 continuity.

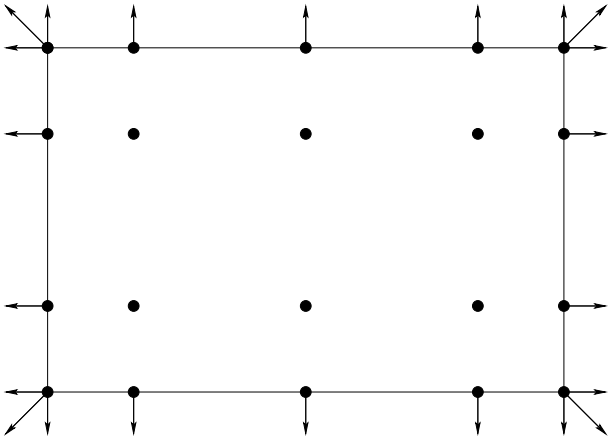
For this, we need

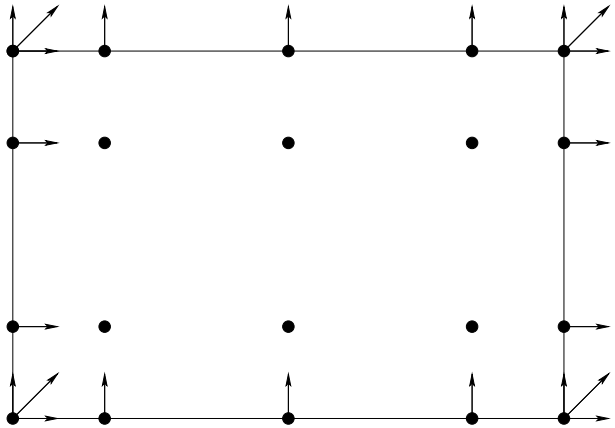
- appropriate definitions of the degrees of freedom
- a layout or ordering of the degrees of freedom into a array, and,
- if possible, fast solvers and fast matrix-vector multiplication.

C1 continuous elements - Degrees of freedom, geometry

Function value: filled circles. Normal derivatives: horizontal and vertical arrows. Mixed second derivatives: diagonal arrows. (Spectral generalization of BFS: Bogner-Fox-Smith.)

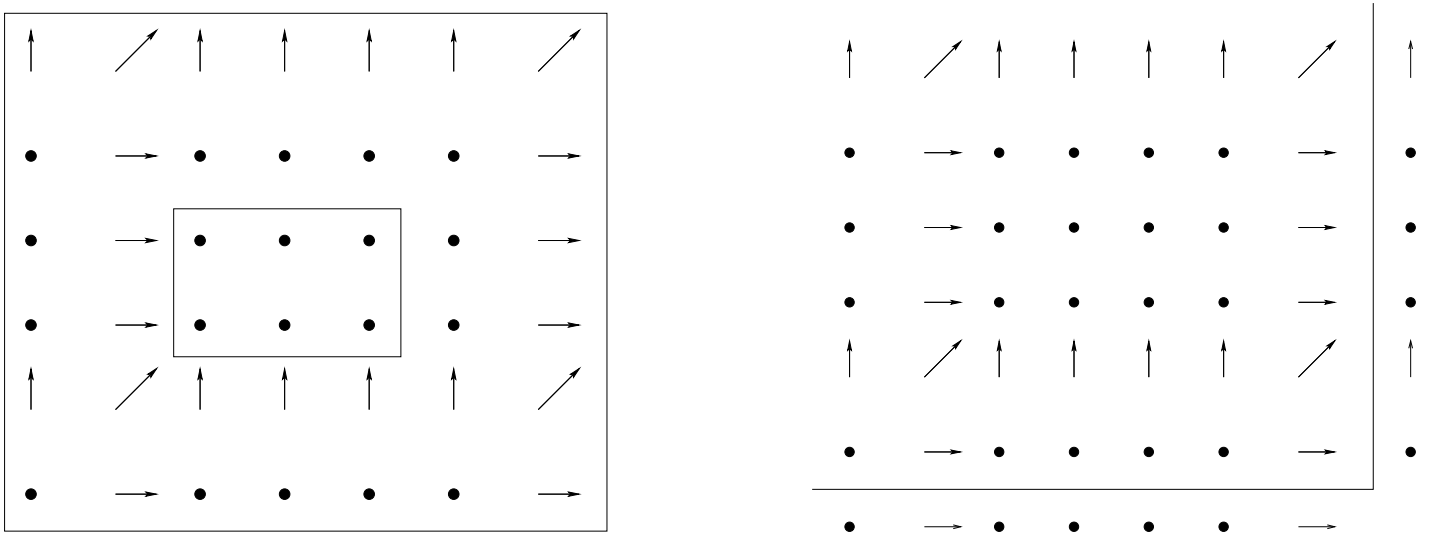
On the right: Reparametrize in coordinate derivatives, horizontal arrows: u_x, vertical arrows: u_y, diagonal arrows: u_{xy}.





C1 continuous elements - Degrees of freedom, arrays

The degrees of freedom just shown fit into an array (left), and the array can be written as a tensor product representation of two one-dimensional representations (right). One element is shown on both sides, and the inner rectangle on the left surrounds the interior degrees of freedom.



C^1 continuous elements - computational aspects

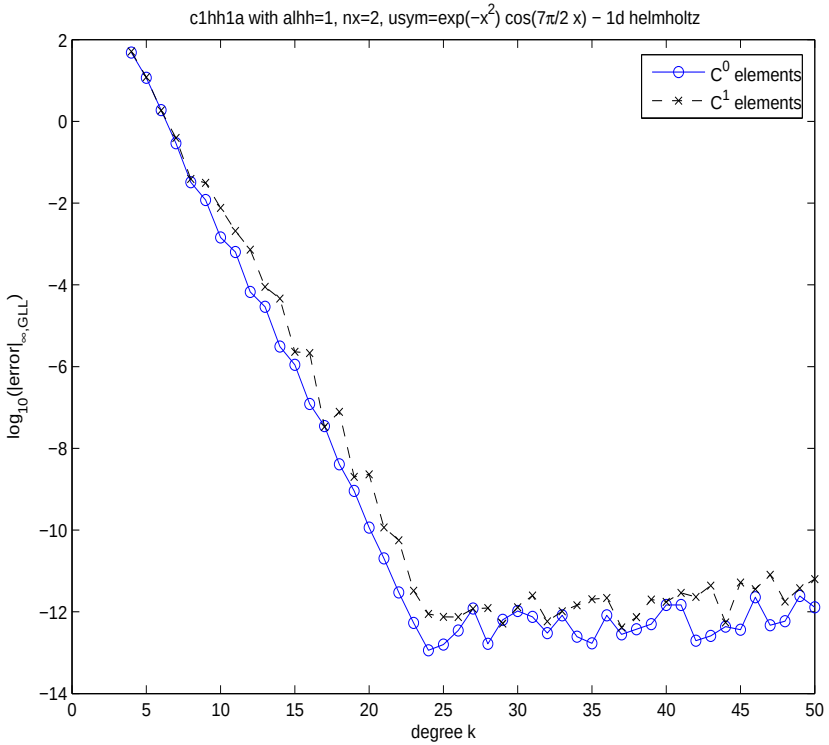
- We work on rectangles or mapped rectangles. For rectangles, use the degrees shown. Start with one-dimensional degrees of freedom. Taking tensor products of these gives us degrees of freedom involving u_x, u_y on the edges, and u_{xy} in the corners.

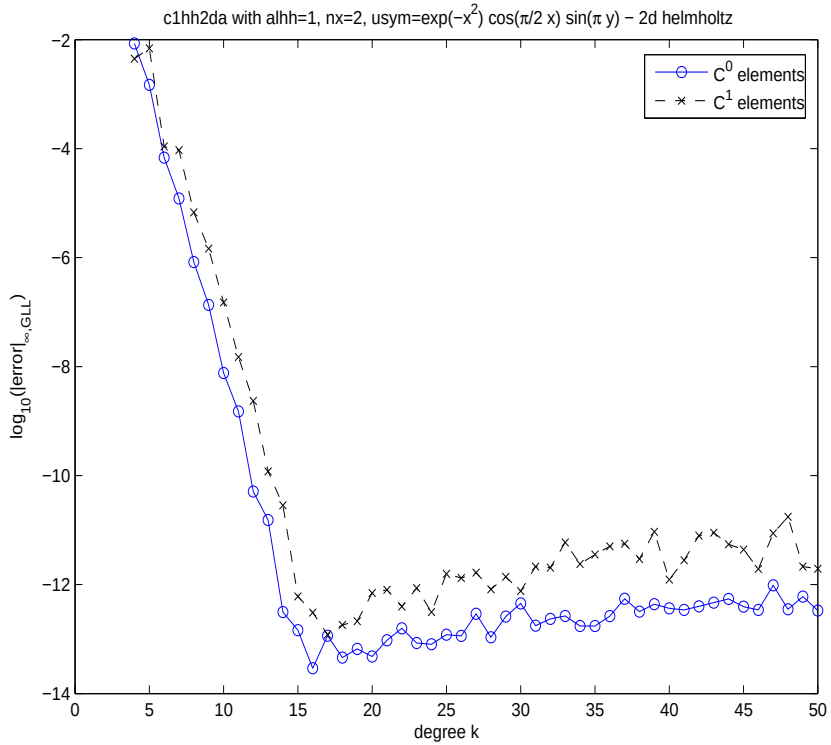
- Easy transformation to normal spectral element degrees of freedom on each element. Can use the same machinery. For rectangle split into regular array of elements, subassembled form for Laplace and Helmholtz equation is still generalized Sylvester equation. Same fast solver.

- On mapped elements, use either coordinate derivative or normal derivatives on edges, and coordinate second derivative in the corner. Degrees of freedom on the edges and corners transform and map in a different way, different structure, solver more involved.

C^1 continuous elements for Helmholtz equation

Left: one-dimensional example. Right: two-dimensional example.





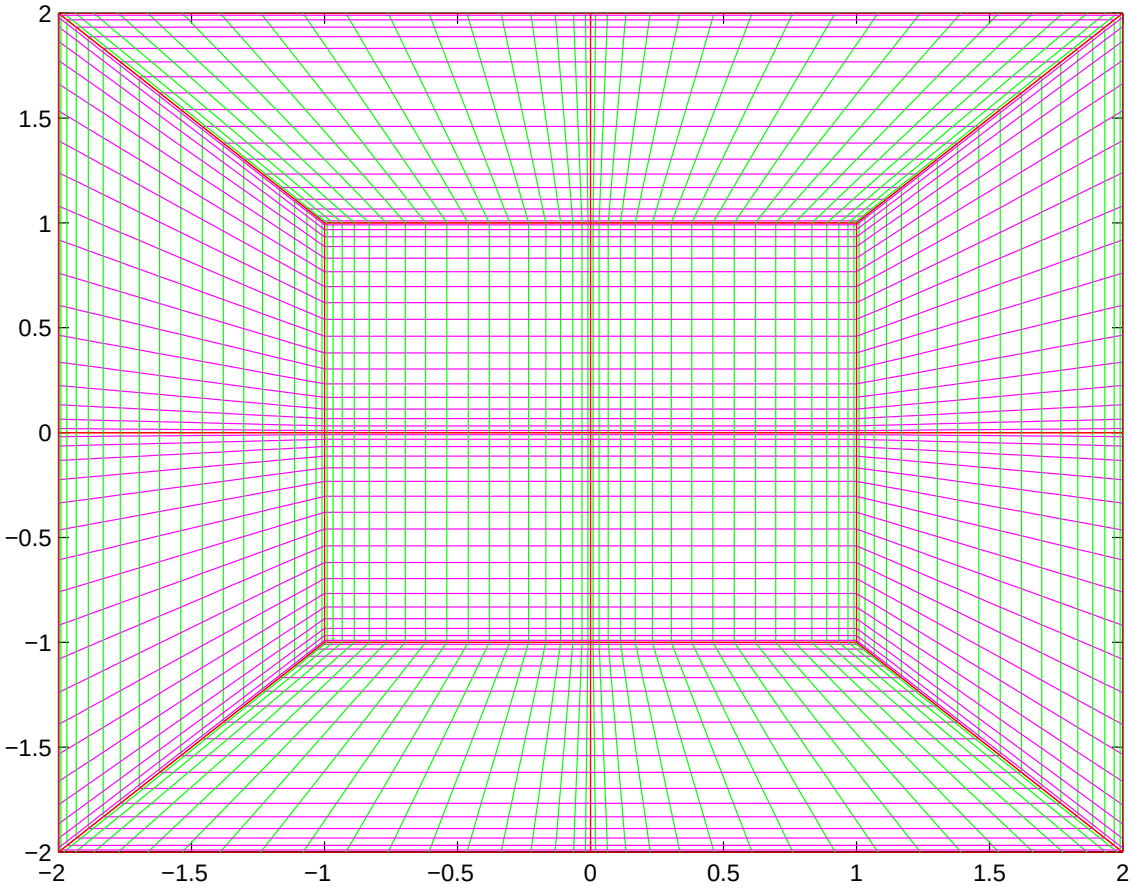
Mapped elements

- Straight-line quadrilaterals require bilinear element mappings, general isoparametric elements represent mapping in spectral element basis. With these mappings, Jacobian and all needed derivatives of (inverse) mapping can be written as combinations of tensor-product matrices (corresponding to matrix-matrix multiplication) and diagonal matrices (corresponding to element-wise multiplication).

- Application of these matrices, and also of the stiffness and mass matrix, to a vector is therefore fast and runs close to peak, even though somewhat more involved. Element-wise preconditioners are also more or less straightforward.

- Solution by static condensation (computing the Schur complement with respect to the interface variables) and then solving the Schur complement system with some standard solver such as SuperLU.

Mapped element layouts



Idea: Mapped lower or higher order C^1 elements

- Maybe there is a way to do it with less pain and effort
- Could use automatic derivation for the PDEs and bilinear forms.
- Could use automatic assembly of element matrices (FIAT: Kirby et al)
- Optimization of assembly and application of element matrices (Knepley et al)
- FIAT (written in PYTHON) should be able to do once one can write down the bilinear form.
- The optimization would also have to be set up for our problem.
- The examples they did with such an approach were Helmholtz/Laplace/Navier-Stokes. MHD and Maxwell don't seem impossible ...

Implementation in C

- Simple idea: code all the matrix operations and algorithms that are already written in high-level linear algebra form in my MATLAB code now in C using LAPACK, BLAS and other libraries as needed.

- Compilation and optimization: done for my laptops and some computers, not all. Full range of needed basic operations is implemented, so that writing new, additional code starts to become more like writing matlab scripts.

- C version runs for the rectangular case on my laptop and on a number of machines at NYU now. Code basis is essentially complete.

- Mapped elements: in active development. C^1-elements: debugging.

- Cross-validated C version of rectangular case against MATLAB.

The package: features, tools, visualization

- Command line interface, some integration with shell/python. (Runs as batch, not interactive).
- Programs store data on GLL grids in simple binary format . Can restart computation from saved data.
- Can interpolate GLL data to uniform grid to produce colorplots (different colormaps and formats) or contour plots (using gnuplot). Can produce MPEG movies from data using these scripts.
- Lots of data for long movie or long integration time: how and what to store? (Make movie in background, throw away data?)
- Have some diagnostics and tools such as spectral expansion, jumps in derivatives, energy, ... that compute data on each element.

Observations

- With appropriate libraries and optimization, C version is much faster, especially on my laptop and newer machines.
- The long story of edge effects for some examples: Different kinds of filtering are successfull, L^2 projection into the C^1 removes effects, but blurs results. Various other filterings (Fischer, lowpass) do improve the results, and high enough resolution and/or appropriate combination of integration and filtering remove the effects.
- Poisson brackets: Higher order integration makes a difference and give results with sharper and finer strutures. In most circumstances, increasing the order weakens the edge effects. However, sometimes higher order elements and/or filtering is needed to completely remove them.
- C^1 results from MATLAB look good in regular case, needs smaller degrees to look good.

Current and future work

- Other examples: still mostly turning mode problem. Working to implement tearing mode, coalescence, Kelvin-Helmholtz,...
- Developping mapped element version as far as rectangular element version, turn working MATLAB into C (by rewriting the code).
- Generate results for the different version and examples. Can run lots of things automatically, including processing of data. Unfortunately cannot automatize looking at it and making sense of it ...
- Move toward inclusion of some modules into M3D.
- Parallelization.