Semi-Lagrangian Methods for [illegible] Intrepid

Scott Moe

Sandia National Laboratories

TUG, Sandia National Labs
Oct. 29, 2014

The Lagrangian View of The Scalar Transport Equations

$$\frac{\partial q}{\partial t} + \mathbf{u} \cdot \nabla q = 0 \quad \text{or} \quad \frac{\partial q}{\partial t} + \cdot(\mathbf{u} q) = 0 \tag{1}$$

Assume $\nabla \cdot \mathbf{u} = 0$ for simplicity

$$\frac{d}{dt} \equiv \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla \tag{2}$$

$$\frac{dq}{dt} = 0 \quad \text{if} \quad \frac{\partial \mathbf{x}}{\partial t} = \mathbf{u} \tag{3}$$

Transport viewed this way is just an ODE describing how points in space move.

Gauss Lobatto Legendre Nodes

- Use $\ell_i(x)$, interpolating polynomials associated with the Gauss Lobatto Legendre Nodes (GLL).
- the $n+1$ GLL nodes exactly integrate $2n-1$ degree polynomials.
- Perform Integration approximately using the GLL nodes
- $\int_{[-1,1]} \ell_i(x)\ell_j(x)dx \approx \omega_i\delta_{ij}$

- The 1-D GLL Quadrature Rule includes interval endpoints enforcing continuity between cells

Points	Weights
-1	$\frac{1}{6}$
$-\sqrt{\frac{1}{5}}$	$\frac{5}{6}$
$\sqrt{\frac{1}{5}}$	$\frac{5}{6}$
1	$\frac{1}{6}$

The Spectral Element Discretization in Two Dimensions

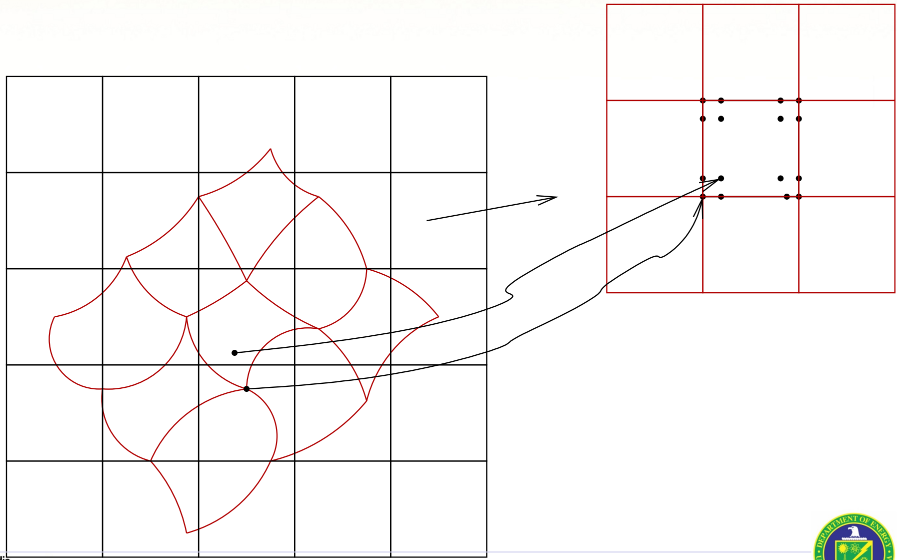
- In two dimensions the cells will be quads
- The basis functions will be tensor products of the 1-d basis functions $\phi_k(x,y) = \ell_{i_k}(x)\ell_{j_k}$
- Define $m_l = \det(J(x_l, y_l))$
- $$M_{l,m} = (\phi_l, \phi_m) = (\ell_{i_l}\ell_{j_l}, \ell_{i_m}\ell_{j_m}) \approx \omega_{i_l}\omega_{j_l} m_l \delta_{lm} \tag{4}$$
- The Spectral Element discretization gives a Continuous Galerkin method that has an approximately diagonal mass-matrix
- The approximation does not affect Formal Order of Accuracy

The Semi-Lagrangian Spectral Element Method

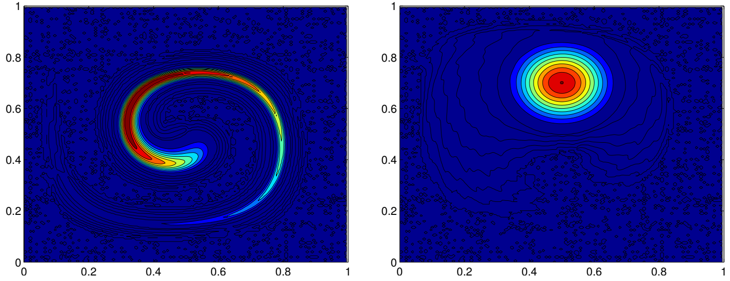
At Each Time-step:

- Start with an initial condition $q(\mathbf{x}, t^n)$
- For each tensor product GLL node in every cell k define $\mathbf{x}_{ij}^k$
- solve $\frac{\partial \mathbf{x}}{\partial t} = \mathbf{u}$ such that $\mathbf{x}(t^{n+1}) = \mathbf{x}_{ij}^k$
- Thus the solution at $q(\mathbf{x}_{ij}^k, t^{n+1}) \approx q(\mathbf{x}, t^n)$

The Semi-Lagrangian Spectral Element Method



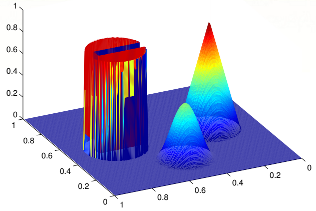
$$u = \sin(\pi x)\sin(\pi x)\sin(2\pi y)\cos(\pi \frac{t}{5})$$
$$v = -\sin(\pi y)\sin(\pi y)\sin(2\pi x)\cos(\pi \frac{t}{5})$$



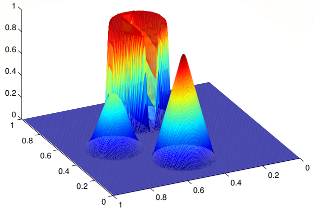
(a) $t = 2.5$ (b) $t = 5.0$ (returns to the initial condition)

Figure: Example with a flow field that is very deformational. The initial profile is a cosine bell, a C_0 function.

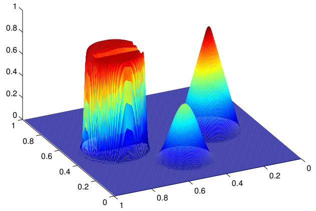
$$u = \left(\frac{1}{2} - y\right), \qquad v = \left(x - \frac{1}{2}\right)$$



(a) $t = 0$



(b) $t = \frac{\pi}{2}$



(c) $t = 2\pi$

What Do We Need?

For this algorithm we only need these operations:

- Locating points in Cells
- Mapping From Physical Cells to a Reference Element
- Interpolation
- Integration
- This is very simple so it should be very efficient!

Computational Efficiency

Table: Comparison to a ALE Finite Volume Method on the Solid Body Rotation Example

Method	Time(Sec)	L2 error
FV 150 DOFs	28.52	7.05×10^{-02}
cubic SEM 150 DOFs equal dt	17.58	7.21×10^{-02}
cubic SEM 150 DOFs equal CFL	5.38	5.83×10^{-02}

- A more thorough analysis of the efficiency as you increase order is a work in progress.

- Define a standard Quadrilateral cell topology

```
shards::getCellTopologyData<Quadrilateral< 4 > >()
CellTopology  quad_4(shards::getCellTopologyData...
                             <Quadrilateral< 4 > >() );
int numNodes=quad_4.getNodeCount();
int spaceDim=quad_4.getDimension();
```

- We have many quad cells, and each cell has $(n+1)^2$ degrees of freedom
- We have an Eularian and Lagrangian Grid
- The Lagrangian Grid nodes will be called Trace-back Points

```
FieldContainer<double> quadNodes(numCells, ...
                           numNodes, spaceDim);
FieldContainer<double> TraceBackPoints(numDofs, spaceDim);
```

- checkPointwiseInclusion can test if points are included in a given cell

```
FieldContainer <int> testPoints(numDofs);
CellTools::checkPointwiseInclusion(testPoints,...
                TraceBackPoints, quadNodes, quad_4,CellNum);
```

- mapToReferenceFrame performs an iterative scheme to find points in the reference coordinates of a given Cell

```
FieldContainer <double> refPoints(numDofs, spaceDim);
CellTools::mapToReferenceFrame(refPoints,...
               TraceBackPoints, quadNodes, quad_4, whichCell);
```

- Interpolate to find values at the reference points in 1d
- POINTTYPE_SPECTRAL refers to Gauss-Lobatto Nodes...you can also interpolate using other sets of nodes like uniformly spaced
- OPERATOR_VALUE could also be OPERATOR_GRAD for Gradients for example.

```
Basis_HGRAD_LINE_Cn_FEM <double, FieldContainer...
                <double>>lineHGradBasis(deg,...
                POINTTYPE_SPECTRAL);
FieldContainer<double> PtEval(numFields,numPoints);
lineHGradBasis.getValues(PtEval,refPoints,...
                OPERATOR_VALUE);
```

- Interpolate to find values at the reference points in 2d

```
Basis_HGRAD_QUAD_Cn_FEM<double, FieldContainer...
                <double> > quadHGradBasis(deg,...
                POINTTYPE_SPECTRAL);
FieldContainer<double> PtEval(numFields, numPoints);
quadHGradBasis.getValues(PtEval,refPoints,...
                OPERATOR_VALUE);
```

How to Perform the SL method

- For each tensor product GLL node in every cell k define $\mathbf{x}_{ij}^k$
- solve $\frac{\partial \mathbf{x}}{\partial t} = \mathbf{u}$ such that $\mathbf{x}(t^{n+1}) = \mathbf{x}_{ij}^k$
- use checkPointwiseInclusion to locate nodes within cells
- use mapToReferenceFrame and quadHGradBasis.getValues to find $q(\mathbf{x}, t^n)$
- Thus the solution at $q(\mathbf{x}_{ij}^k, t^{n+1}) \approx q(\mathbf{x}, t^n)$

- Find the Determinant of the Jacobian at each Cubature point...this is necessary to do things like integrate the global solution
- Define 1d Gauss-Lobatto Legendre Cubature Operator.

```
CellTools::setJacobian(refQuadJacobian, cubPoints,...
                    QuadNodes, quad_4);
CellTools::setJacobianDet(refQuadJacobDet, refQuadJacobian );
```

- There are many quadrature and cubature rules built into Trilinos that you can access (Gauss rules, Gauss-Lobatto rules, Gauss-Radau rules..)

```
Teuchos::RCP<Cubature<double,FieldContainer<double>,...
                    FieldContainer<double>>> glcub
= Teuchos::rcp(new CubaturePolylib<double,...

FieldContainer<double>,FieldContainer<double>>...
                    (max(2n − 1,0),PL_GAUSS_LOBATTO) );
```

- From the cubature operator get the 1d Gauss-Lobatto-Legendre cubature points and weights

```
int numCubPoints = glcub→getNumPoints();
FieldContainer<double> cubPoints1D(np, 1);
FieldContainer<double> cubWeights1D(np);
glcub→getCubature(cubPoints1D,cubWeights1D);
```