SAND2009-7358P

Capability Couplings with Intrepid

Trilinos User's Group Meeting

November 4, 2009

Kara Peterson

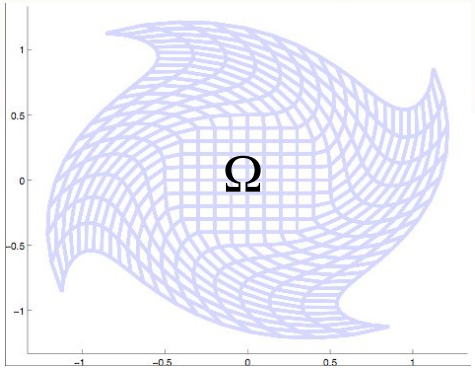
Pavel Bochev
David Hensinger
Denis Ridzal
Chris Siefert

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

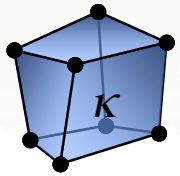
SAND2009-7358P

Solving PDEs with FEM in Trilinos

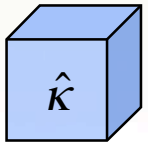
Pamgen: Mesh



Shards: Ref. Cell Topology



$F : \hat{\kappa} \rightarrow \kappa$



Intrepid: Transform, Integration, Discrete Spaces and Operators

$$\mathbf{K}^{e}_{ij} = \int_{\kappa} \nabla\phi_i(x)\nabla\phi_j(x)dx$$

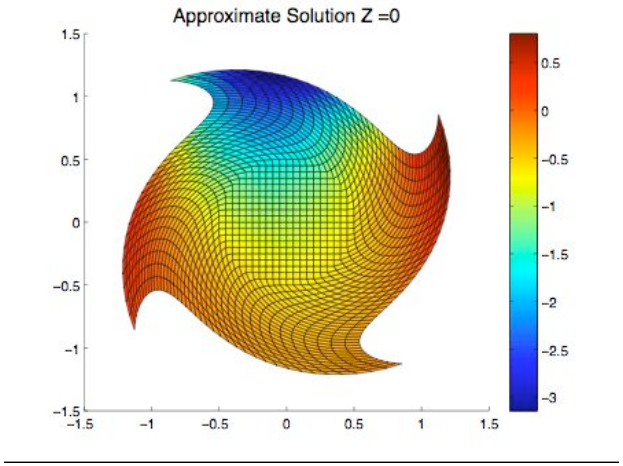
$$\mathcal{L}u = f \text{ in } \Omega$$

Epetra: Global Matrices

$$\mathbf{K}\, u_h = \mathbf{b}$$

ML: Solve

$$u_h = \mathbf{K}^{-1}\, \mathbf{b}$$



Example Drivers

Poisson Equation with H(grad) Elements

$$\begin{cases} \Delta u = f & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases}$$

→ example_Poisson, example_03

LSFEM Div-Curl System with H(curl) Elements

$$\begin{cases} \nabla \times \mathbf{u} = \mathbf{f} & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = g & \text{in } \Omega \\ \mathbf{n} \times \mathbf{u} = 0 & \text{on } \partial\Omega \end{cases}$$

→ example_CurlLSFEM, example_01

LSFEM Div-Curl System with H(div) Elements

$$\begin{cases} \nabla \times \mathbf{u} = \mathbf{f} & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = g & \text{in } \Omega \\ \mathbf{n} \cdot \mathbf{u} = 0 & \text{on } \partial\Omega \end{cases}$$

→ example_DivLSFEM, example_02

Located in:

- Trilinos/packages/trilinoscouplings/examples/scaling
- Trilinos/packages/intrepid/example/Drivers

Functionality

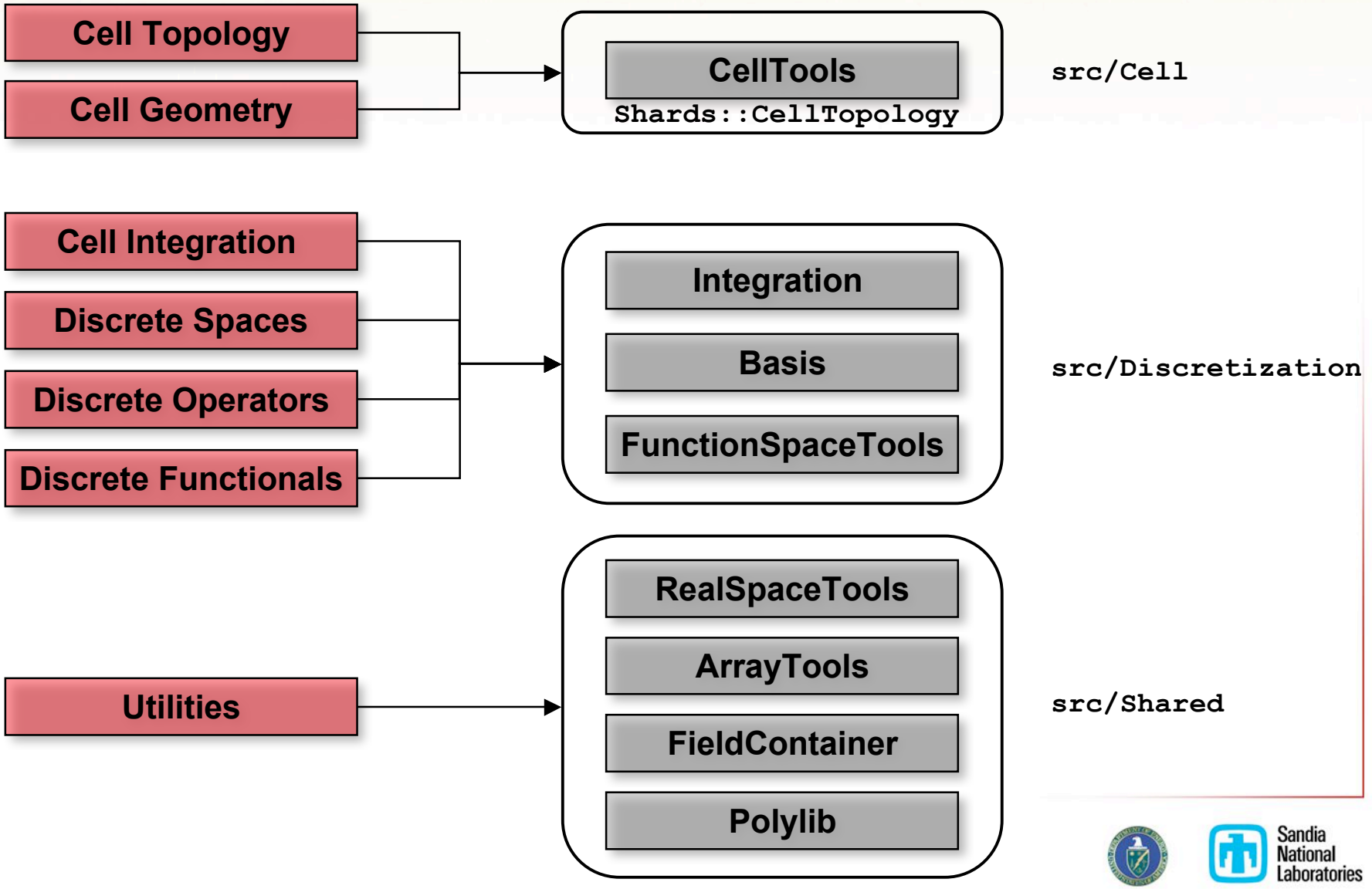
- Intrepid for element matrix assembly
- Epetra for global matrices
- PAMGEN for meshing
- ML to solve
- Intrepid for error estimates
- Runs on multiple processors
- Intrepid for element matrix assembly
- Epetra for global matrices

Current Status of Intrepid

Released with Trilinos 10.0: a suite of "mathematical" (stateless) tools for

- **Cell topology**
  - Topology type (base vs. extended), cell type (standard = has reference cell vs. non-standard), adjacency queries, permutations,…
- **Cell geometry**
  - Jacobians, maps to and from reference cells (if applicable), surface normal, line tangent, subcell measure,…
- **Cell integration**
  - Cubature points on standard cells, direct integration on non-standard cells
- **Discrete spaces on a cell workset**
  - Definition of finite dimensional approximation spaces, reconstruction (interpolation) operators, transformation rules,…
- **Discrete operators on a cell workset**
  - Basic differential operators acting on scalar, vector and tensor fields
- **Discrete functionals on a cell workset**
  - Basic linear functionals acting on scalar, vector and tensor fields
- **Utilities**
  - Everything else you may need but that does not fit in any of the above categories

Mapping Functionality to Software



Intrepid Example: Compatible LSFEM for div-curl Systems

A model Div-Curl System

$$\begin{cases} \nabla \times \mathbf{u} = \mathbf{f} & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = g & \text{in } \Omega \\ \mathbf{n} \times \mathbf{u} = 0 & \text{on } \partial\Omega \end{cases}$$

$\Omega \subset \mathbf{R}^3$ → bounded contractible domain

$\partial\Omega$ → Lipschitz continuous

A continuous least-squares principle

$$\begin{cases} J(\mathbf{u};\mathbf{f},g) = \|\nabla \times \mathbf{u} - \mathbf{f}\|_0^2 + \|\nabla \cdot \mathbf{u} - g\|_0^2 \\ X = H_0(curl,\Omega) \cap H(div,\Omega) \end{cases} \Rightarrow \min_X J(\mathbf{u};\mathbf{f},g)$$

A semi-conforming LSFEM:

$$\begin{cases} J(\mathbf{u}_h;\mathbf{f},g) = \|\nabla \times \mathbf{u}_h - \mathbf{f}\|_0^2 + \|\nabla_h^* \cdot \mathbf{u}_h - g\|_0^2 \\ X_h \subset H_0(curl,\Omega) \cap \cancel{H(div,\Omega)} \end{cases}$$

$X_h = \mathbf{C}_0^h$ is curl-conforming FE, e.g., Nedelec

We gain some and loose some:

- curl-conforming FE: natural for the tangential boundary condition
- curl-conforming FE: not in the domain of div - need discrete approximation!

From Discretization to Linear System

For clarity we now focus on the lowest-order case

$\mathbf{C}_0^h \rightarrow$ the lowest-order Nedelec space of the 1st kind

$G_0^h \rightarrow$ the lowest-order nodal C^0 space

Discrete least-squares problem

$$\left(\nabla\times\mathbf{u}_h,\nabla\times\mathbf{v}_h\right)_0+\left(\nabla_h^*\cdot\mathbf{u}_h,\nabla_h^*\cdot\mathbf{v}_h\right)_0=\left(\mathbf{f},\nabla\times\mathbf{v}_h\right)_0+\left(g,\nabla_h^*\cdot\mathbf{v}_h\right)_0 \quad \forall\mathbf{v}_h\in\mathbf{C}_0^h$$

$$(\mathbf{K} + \mathbf{M}_C\mathbf{D}_{VE}\mathbf{M}_G^{-1}\mathbf{D}_{VE}^T\mathbf{M}_C)\ \mathbf{u}_h = \mathbf{b}$$

curl-curl + **grad-div** matrices

where

- $\mathbf{M}_C$ **curl-conforming mass matrix**
- $\mathbf{M}_G$ **grad-conforming mass matrix**
- $\mathbf{D}_{VE}$ **vertex-to-edge incidence matrix**

Note: Can use any *O(h)* approximation for $\mathbf{M}_G$
We will use mass lumping.

- We **already have** an AMG solver for this system in Trilinos - a subsolver of the eddy current solver developed in *Bochev, Hu, Tuminaro and Siefert, SISC 2008.*
- The ML solver requires 4 matrices: $\mathbf{K}$, $\mathbf{M}_C$, $\mathbf{M}_G$, and $\mathbf{D}_{VE}$

Using Intrepid to Assemble the System

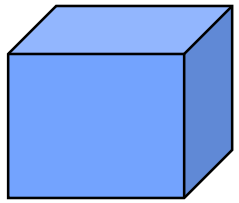
Procedure

1. **Set up**: define cell, integration method (cubature), and reference cell basis
   a. Choose cell type: **`Hexahedron<8>`** and generate grid (PAMGEN)
   b. Define **`Cubature`** on reference cell and on a typical face (for B.C.)
   c. Define & evaluate reference cell H(curl) basis: **`Basis_HCURL_HEX_I1_FEM`**
   d. Define & evaluate reference cell H(grad) basis: **`Basis_HGRAD_HEX_C1_FEM`**
2. **Set up:** get transformation data
   a. Define a physical cell workset
   b. Define multi-dimensional arrays for cell data (`FieldContainer<double>`)
   c. Get pullback data at cubature points (Jacobians, determinants, face area…)
   d. Get basis signs (=edge permutations) for H(curl) basis
3. **Assemble**
   a. Assemble the H(grad) mass matrix
   b. Assemble the H(curl) mass matrix
   c. Assemble the H(curl) stiffness matrix
   d. Generate the incidence matrix
   e. Assemble right-hand side
4. **Solve:** using ML

Step 1: Define cell, cubature, and basis

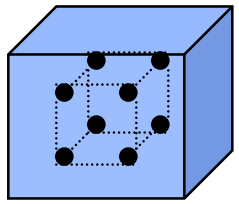
Define Cell Topology with Shards

```
typedef shards::CellTopology    CellTopology;
CellTopology hex_8(shards::getCellTopologyData<Hexahedron<8> >() );
int numNodesPerCell = hex_8.getNodeCount();
int numEdgesPerCell = hex_8.getEdgeCount();
int spaceDim = hex_8.getDimension();
```



Get Cubature Points and Weights

```
DefaultCubatureFactory<double>  cubFactory;
int cubDegree = 2;
Teuchos::RCP<Cubature<double> > hexCub = cubFactory.create(hex_8, cubDegree);
int cubDim       = hexCub->getDimension();
int numCubPoints = hexCub->getNumPoints();

FieldContainer<double> cubPoints(numCubPoints, cubDim);
FieldContainer<double> cubWeights(numCubPoints);
hexCub->getCubature(cubPoints, cubWeights);
```



Sandia National Laboratories

Step 1: Define cell, cubature, and basis

Define Basis on reference cell

```
Basis_HCURL_HEX_I1_FEM<double, FieldContainer<double> > hexHCurlBasis;
Basis_HGRAD_HEX_C1_FEM<double, FieldContainer<double> > hexHGradBasis;
int numFieldsC = hexHCurlBasis.getCardinality();
int numFieldsG = hexHGradBasis.getCardinality();
```

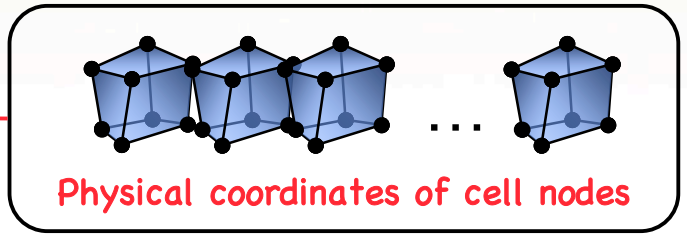
Evaluate Basis at Cubature Points

```
FieldContainer<double> hexGVals(numFieldsG, numCubPoints);
FieldContainer<double> hexCVals(numFieldsC, numCubPoints, spaceDim);
FieldContainer<double> hexCurls(numFieldsC, numCubPoints, spaceDim);

hexHGradBasis.getValues(hexGVals, cubPoints, OPERATOR_VALUE);
hexHCurlBasis.getValues(hexCVals, cubPoints, OPERATOR_VALUE);
hexHCurlBasis.getValues(hexCurls, cubPoints, OPERATOR_CURL);
```

Step 2: Pullback data: Jacobians

Define physical cell workset

```
// Intialize hexWorkset from mesh data
int numCells = 100;
FieldContainer<double> hexWorkset(numCells,
                                  numNodesPerCell,
                                  spaceDim);
```



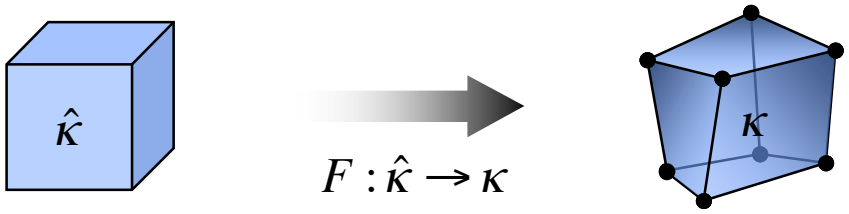
Compute Jacobians of cell workset

```
FieldContainer<double> hexJacobian(numCells, numCubPoints, spaceDim, spaceDim);
FieldContainer<double> hexJacobInv(numCells, numCubPoints, spaceDim, spaceDim);
FieldContainer<double> hexJacobDet(numCells, numCubPoints);

CellTools::setJacobian(hexJacobian, cubPoints, hexWorkset, hex_8);
CellTools::setJacobianInv(hexJacobInv, hexJacobian );
CellTools::setJacobianDet(hexJacobDet, hexJacobian );
```

$\rightarrow DF$

$\rightarrow DF^{-1}$

$\rightarrow \det DF$

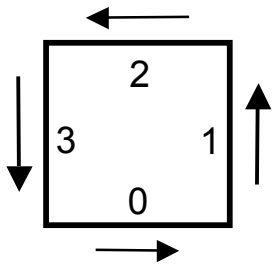


Sandia National Laboratories

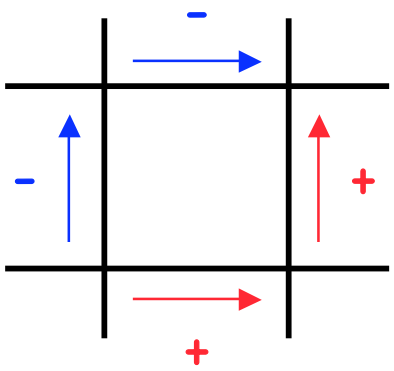
Step 2: Pullback data: H(curl) basis signs

```
FieldContainer<double> hexBasisSigns(numCells, numFieldsC);
```

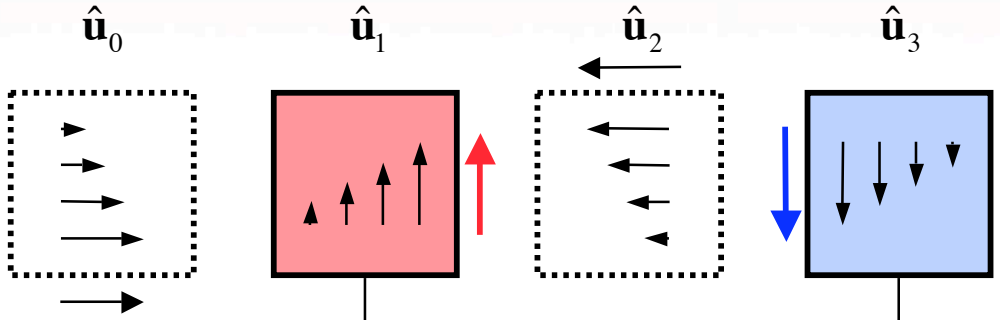
Local (reference) cell



Orientations



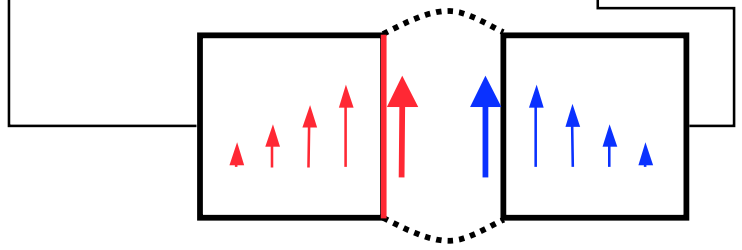
Global (physical) cell



Basis functions

$$\mathbf{u}_1\big|_{K_L} = +\Phi^*(\hat{\mathbf{u}}_1)$$

$$\mathbf{u}_3\big|_{K_R} = -\Phi^*(\hat{\mathbf{u}}_3)$$



$$\mathbf{u}_i\big|_K = \sigma_{i(K)}\Phi^*(\hat{\mathbf{u}}_{i(K)})$$

Step 3: Assembly

H(grad) mass matrix

$$\left(\mathbf{M}_G\right)_{ij} = \int_{\Omega} \phi_i(x)\phi_j(x)dx = \int_{\Omega} (\Phi^* \hat{\phi}_i)(x)(\Phi^* \hat{\phi}_j)(x)dx = \int_{\hat{\Omega}} \hat{\phi}_i(\hat{x})\hat{\phi}_j(\hat{x})J(\hat{x})d\hat{x} \approx \sum_{p=1}^{numPt} \hat{\phi}_i(\hat{x}_p)\hat{\phi}_j(\hat{x}_p)J(\hat{x}_p)\omega_p$$

H(curl) mass matrix

$$\left(\mathbf{M}_C\right)_{ij} = \int_{\Omega} \mathbf{u}_i(x)\cdot\mathbf{u}_j(x)dx = \int_{\Omega} (\Phi^* \hat{\mathbf{u}}_i)(x)\cdot(\Phi^* \hat{\mathbf{u}}_j)(x)dx$$

$$= \int_{\hat{\Omega}} \sigma_i\sigma_j\left(DF^{-T}\hat{\mathbf{u}}_i(\hat{x})\right)\cdot\left(DF^{-T}\hat{\mathbf{u}}_j(\hat{x})\right)J(\hat{x})d\hat{x} \approx \sum_{p=1}^{numPt} \sigma_i\sigma_j\left(DF^{-T}\hat{\mathbf{u}}_i(\hat{x}_p)\right)\cdot\left(DF^{-T}\hat{\mathbf{u}}_j(\hat{x}_p)\right)J(\hat{x}_p)\omega_p$$

H(curl) stiffness matrix

$$\left(\mathbf{K}_C\right)_{ij} = \int_{\Omega} \nabla\times\mathbf{u}_i(x)\cdot\nabla\times\mathbf{u}_j(x)dx = \int_{\Omega} \left(\Phi^*(\hat{\nabla}\times\hat{\mathbf{u}}_i)\right)(x)\cdot\left(\Phi^*(\hat{\nabla}\times\hat{\mathbf{u}}_j)\right)(x)dx$$

$$= \int_{\hat{\Omega}} \sigma_i\sigma_j\left(J^{-1}DF(\hat{\nabla}\times\hat{\mathbf{u}}_i)(\hat{x})\right)\cdot\left(J^{-1}DF(\hat{\nabla}\times\hat{\mathbf{u}}_j)(\hat{x})\right)J(\hat{x})d\hat{x}$$

$$\approx \sum_{p=1}^{numPt} \sigma_i\sigma_j\left(J^{-1}DF(\hat{\nabla}\times\hat{\mathbf{u}}_i)(\hat{x}_p)\right)\cdot\left(J^{-1}DF(\hat{\nabla}\times\hat{\mathbf{u}}_j)(\hat{x}_p)\right)J(\hat{x}_p)\omega_p$$

Sandia National Laboratories

Step 3: Assembly

Right hand side

$$
\begin{aligned}
\left(\mathbf{b}\right)_i &= \int_{\Omega} \nabla \times \mathbf{u}_i(x) \cdot \mathbf{f}(x) dx + \int_{\Omega} \nabla^* \cdot \mathbf{u}_i(x) g(x) dx \\
&= \int_{\Omega} \nabla \times \mathbf{u}_i(x) \cdot \mathbf{f}(x) dx - \int_{\Omega} \mathbf{u}_i(x) \cdot \nabla g(x) dx + \int_{\Gamma} g(x) \mathbf{u}_i(x) \cdot \mathbf{n} d\Gamma \\
&= \int_{\hat{\Omega}} \sigma_i (J^{-1} DF(\nabla \times \hat{\mathbf{u}}_i)(\hat{x})) \cdot \mathbf{f}(\hat{x}) J(\hat{x}) d\hat{x} - \int_{\hat{\Omega}} \sigma_i DF^{-T} \hat{\mathbf{u}}_i(\hat{x}) \cdot \nabla g(\hat{x}) J(\hat{x}) d\hat{x} + \int_{\hat{\Gamma}} g(\hat{x}) \sigma_i DF^{-T} \hat{\mathbf{u}}_i(\hat{x}) \cdot \mathbf{n} d\hat{\Gamma} \\
&\approx \sum_{p=1}^{numPt} \sigma_i \left(J^{-1} DF(\nabla \times \hat{\mathbf{u}}_i)(\hat{x}_p) \right) \cdot \left(\mathbf{f}(\hat{x}_p) \right) J(\hat{x}_p) \omega_p - \sum_{p=1}^{numPt} \sigma_i \left(DF^{-T}(\hat{\mathbf{u}}_i)(\hat{x}_p) \right) \cdot \left(\nabla g(\hat{x}_p) \right) J(\hat{x}_p) \omega_p \\
&\quad + \sum_{f=1}^{nBndF} \sum_{fp=1}^{numFPt} \sigma_i \left(DF^{-T} \hat{\mathbf{u}}_i(\hat{x}_{fp}) \right) \cdot \mathbf{n}_f \left(g(\hat{x}_{fp}) \right) J(\hat{x}_{fp}) \omega_{fp}
\end{aligned}
$$

Step 3: Declare arrays

```
// Containers for element HGRAD mass matrix
FieldContainer<double> massMatrixG(numCells, numFieldsG, numFieldsG);
FieldContainer<double> weightedMeasure(numCells, numCubPoints);
FieldContainer<double> weightedMeasureMuInv(numCells, numCubPoints);
FieldContainer<double> hexGValsTrans(numCells, numFieldsG, numCubPoints);
FieldContainer<double> hexGValsTransWeighted(numCells, numFieldsG, numCubPoints);
```

```
// Containers for element HCURL mass matrix
FieldContainer<double> massMatrixC(numCells, numFieldsC, numFieldsC);
FieldContainer<double> hexCValsTrans(numCells, numFieldsC, numCubPoints, spaceDim);
FieldContainer<double> hexCValsTransWeighted(numCells, numFieldsC, numCubPoints, spaceDim);

// Containers for element HCURL stiffness matrix
FieldContainer<double> stiffMatrixC(numCells, numFieldsC, numFieldsC);
FieldContainer<double> weightedMeasureMu(numCells, numCubPoints);
FieldContainer<double> hexCurlsTrans(numCells, numFieldsC, numCubPoints, spaceDim);
FieldContainer<double> hexCurlsTransWeighted(numCells, numFieldsC, numCubPoints, spaceDim);
```

```
// Global arrays in Epetra format
Epetra_FECrsMatrix MassG(Copy, globalMapG, numFieldsG);
Epetra_FECrsMatrix MassC(Copy, globalMapC, numFieldsC);
Epetra_FECrsMatrix StiffC(Copy, globalMapC, numFieldsC);
Epetra_FEVector rhsC(globalMapC);
```

Step 3: Assemble H(grad) mass matrix

```
fst::HGRADtransformVALUE<double>(hexGValsTrans,
                                 hexGVals);
```

Note that for H(Grad) basis values transformation is formal - simply replicates values in rank-3 array

$\hat{\phi}_i(\hat{x}_p) \rightarrow \hat{\phi}_i(\hat{x}_p)$

Transform to physical coordinates

```
fst::computeCellMeasure<double>(cellMeasure,
                                hexJacobDet,
                                cubWeights);
```

$J(\hat{x}_p)\omega_p \rightarrow J(\hat{x}_p)$, ω_p

Compute cell measure

```
fst::multiplyMeasure<double>(hexGValsTransWeighted,
                             cellMeasure,
                             hexGValsTrans);
```

$\hat{\phi}_j(\hat{x}_p)J(\hat{x}_p)\omega_p \rightarrow J(\hat{x}_p)\omega_p$, $\hat{\phi}_i(\hat{x}_p)$

Multiply basis and cell measure

```
fst::integrate<double>(massMatrixG,
                       hexGValsTrans,
                       hexGValsTransWeighted,
                       COMP_BLAS);
```

$\sum_{p=1}^{numPt} \hat{\phi}_i(\hat{x}_p)\hat{\phi}_j(\hat{x}_p)J(\hat{x}_p)\omega_p \rightarrow \hat{\phi}_i(\hat{x}_p)$, $\hat{\phi}_j(\hat{x}_p)J(\hat{x}_p)\omega_p$

Integrate to compute element mass matrix

$$\left(\mathbf{M}_G\right)_{ij} \approx \sum_{p=1}^{numPt} \hat{\phi}_i(\hat{x}_p)\hat{\phi}_j(\hat{x}_p)J(\hat{x}_p)\omega_p$$

Step 3: Assemble H(curl) mass matrix

```
fst::HCURLtransformVALUE<double>(hexCValsTrans,
                                 hexJacobInv,
                                 hexCVals);

fst::multiplyMeasure<double>(hexCValsTransWeighted,
                reuse →  cellMeasure,
                         hexCValsTrans);

fst::integrate<double>(massMatrixC,
                       hexCValsTrans,
                       hexCValsTransWeighted,
                       COMP_BLAS);

fst::applyLeftFieldSigns<double>(massMatrixC,
                                 hexBasisSigns);

fst::applyRightFieldSigns<double>(massMatrixC,
                                  hexBasisSigns);
```

$DF^{-T}(\hat{x}_p)\hat{\mathbf{u}}_i(\hat{x}_p)$

$DF^{-T}(\hat{x}_p)$

$\hat{\mathbf{u}}_i(\hat{x}_p)$

$DF^{-T}(\hat{x}_p)\hat{\mathbf{u}}_i(\hat{x}_p)J(\hat{x}_p)\omega_p$

$J(\hat{x}_p)\omega_p$

$DF^{-T}(\hat{x}_p)\hat{\mathbf{u}}_i(\hat{x}_p)$

$\sum_{p=1}^{numPt}\left(DF^{-T}\hat{\mathbf{u}}_i(\hat{x}_p)\right)\cdot\left(DF^{-T}\hat{\mathbf{u}}_j(\hat{x}_p)\right)J(\hat{x}_p)\omega_p$

$DF^{-T}(\hat{x}_p)\hat{\mathbf{u}}_i(\hat{x}_p)$

$DF^{-T}(\hat{x}_p)\hat{\mathbf{u}}_i(\hat{x}_p)J(\hat{x}_p)\omega_p$

$\sum_{p=1}^{numPt}\sigma_i\left(DF^{-T}\hat{\mathbf{u}}_i(\hat{x}_p)\right)\cdot\left(DF^{-T}\hat{\mathbf{u}}_j(\hat{x}_p)\right)J(\hat{x}_p)\omega_p$

σ_i

$\sum_{p=1}^{numPt}\sigma_i\sigma_j\left(DF^{-T}\hat{\mathbf{u}}_i(\hat{x}_p)\right)\cdot\left(DF^{-T}\hat{\mathbf{u}}_j(\hat{x}_p)\right)J(\hat{x}_p)\omega_p$

σ_j

$$\left(\mathbf{M}_C\right)_{ij} \approx \sum_{p=1}^{numPt}\sigma_i\sigma_j\left(DF^{-T}\hat{\mathbf{u}}_i(\hat{x}_p)\right)\cdot\left(DF^{-T}\hat{\mathbf{u}}_j(\hat{x}_p)\right)J(\hat{x}_p)\omega_p$$

Sandia National Laboratories

Step 3: Assemble H(curl) stiffness matrix

```
fst::HCURLtransformCURL<double>(hexCurlsTrans,
                                hexJacob,
                                hexJacobianDet,
                                hexCVals);

fst::multiplyMeasure<double>(hexCurlsTransWght,
                reuse →  cellMeasure,
                         hexCurlsTrans);

fst::integrate<double>(stiffMatrixC,
                       hexCurlsTrans,
                       hexCurlsTransWght,
                       COMP_BLAS);

fst::applyLeftFieldSigns<double>(stiffMatrixC,
                                 hexBasisSigns);

fst::applyRightFieldSigns<double>(stiffMatrixC,
                                  hexBasisSigns);
```

$$J^{-1}(\hat{x}_p)DF(\hat{x}_p)(\hat{\nabla}\times\hat{\mathbf{u}}_j)(\hat{x}_p)$$

$$DF(\hat{x}_p)$$

$$J(\hat{x}_p)$$

$$(\hat{\nabla}\times\hat{\mathbf{u}}_j)(\hat{x}_p)$$

$$J^{-1}(\hat{x}_p)DF(\hat{x}_p)(\hat{\nabla}\times\hat{\mathbf{u}}_j)(\hat{x}_p)\,(J(\hat{x}_p)\omega_p)$$

$$J(\hat{x}_p)\omega_p$$

$$J^{-1}(\hat{x}_p)DF(\hat{x}_p)(\hat{\nabla}\times\hat{\mathbf{u}}_j(\hat{x}_p))$$

$$\sum_{p=1}^{numPt}\left(J^{-1}DF(\hat{\nabla}\times\hat{\mathbf{u}}_i)(\hat{x}_p)\right)\cdot\left(J^{-1}DF(\hat{\nabla}\times\hat{\mathbf{u}}_j)(\hat{x}_p)\right)J(\hat{x}_p)\omega_p$$

$$\sigma_i$$

$$\sum_{p=1}^{numPt}\sigma_i\left(J^{-1}DF(\hat{\nabla}\times\hat{\mathbf{u}}_i)(\hat{x}_p)\right)\cdot\left(J^{-1}DF(\hat{\nabla}\times\hat{\mathbf{u}}_j)(\hat{x}_p)\right)J(\hat{x}_p)\omega_p$$

$$\sigma_j$$

$$\sum_{p=1}^{numPt}\sigma_i\sigma_j\left(J^{-1}DF(\hat{\nabla}\times\hat{\mathbf{u}}_i)(\hat{x}_p)\right)\cdot\left(J^{-1}DF(\hat{\nabla}\times\hat{\mathbf{u}}_j)(\hat{x}_p)\right)J(\hat{x}_p)\omega_p$$

$$(\mathbf{K}_C)_{ij}\approx\sum_{p=1}^{numPt}\sigma_i\sigma_j\left(J^{-1}DF(\hat{\nabla}\times\hat{\mathbf{u}}_i)(\hat{x}_p)\right)\cdot\left(J^{-1}DF(\hat{\nabla}\times\hat{\mathbf{u}}_j)(\hat{x}_p)\right)J(\hat{x}_p)\omega_p$$

Step 3: Assemble right hand side

$$(\mathbf{b})_i = \int_\Omega \nabla \times \mathbf{u}_i(x) \cdot \mathbf{f}(x) dx - \int_\Omega \mathbf{u}_i(x) \cdot \nabla g(x) dx + \int_\Gamma g(x) \mathbf{u}_i(x) \cdot \mathbf{n} d\Gamma$$

```
fst::integrate<double>(rhsf,
                       fData,
            reuse →    hexCurlsTransWght,
                       COMP_BLAS);
```

$$\rightarrow \sum_{p=1}^{numPt} \left(J^{-1} DF (\hat{\nabla} \times \hat{\mathbf{u}}_i)(\hat{x}_p) \right) \cdot \left(\mathbf{f}(\hat{x}_p) \right) J(\hat{x}_p) \omega_p$$

$$\leftarrow \mathbf{f}(\hat{x}_p)$$

$$\leftarrow J^{-1}(\hat{x}_p) DF(\hat{x}_p) (\hat{\nabla} \times \hat{\mathbf{u}}_j)(\hat{x}_p) \, (J(\hat{x}_p) \omega_p)$$

```
fst::applyFieldSigns<double>(rhsfData,
                             hexBasisSigns);
```

$$\leftrightarrow \sum_{p=1}^{numPt} \sigma_i \left(J^{-1} DF (\hat{\nabla} \times \hat{\mathbf{u}}_i)(\hat{x}_p) \right) \cdot \left(\mathbf{f}(\hat{x}_p) \right) J(\hat{x}_p) \omega_p$$

$$\leftarrow \sigma_i$$

```
fst::integrate<double>(rhsg,
                       gData,
            reuse →    hexCValsTransWght,
                       COMP_BLAS);
```

$$\rightarrow \sum_{p=1}^{numPt} \left(DF^{-T} \hat{\mathbf{u}}_i(\hat{x}_p) \right) \cdot \left(\nabla g(\hat{x}_p) \right) J(\hat{x}_p) \omega_p$$

$$\leftarrow \nabla g(\hat{x}_p)$$

$$\leftarrow DF^{-T}(\hat{x}_p) \hat{\mathbf{u}}_i(\hat{x}_p) J(\hat{x}_p) \omega_p$$

```
fst::applyFieldSigns<double>(rhsgData,
                             hexBasisSigns);
```

$$\leftrightarrow \sum_{p=1}^{numPt} \sigma_i \left(DF^{-T} \hat{\mathbf{u}}_i(\hat{x}_p) \right) \cdot \left(\nabla g(\hat{x}_p) \right) J(\hat{x}_p) \omega_p$$

$$\leftarrow \sigma_i$$

$$(\mathbf{b})_i \approx \sum_{p=1}^{numPt} \sigma_i \left(J^{-1} DF (\nabla \times \hat{\mathbf{u}}_i)(\hat{x}_p) \right) \cdot \left(\mathbf{f}(\hat{x}_p) \right) J(\hat{x}_p) \omega_p - \sum_{p=1}^{numPt} \sigma_i \left(DF^{-1} (\hat{\mathbf{u}}_i)(\hat{x}_p) \right) \cdot \left(\nabla g(\hat{x}_p) \right) J(\hat{x}_p) \omega_p + B.T.$$

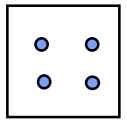
Sandia National Laboratories

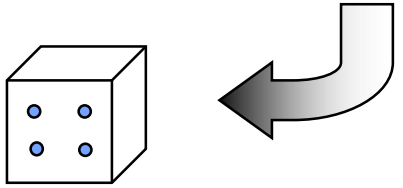
Step 3: Assemble right hand side

$$(\mathbf{b})_i = \int_\Omega \nabla \times \mathbf{u}_i(x) \cdot \mathbf{f}(x) dx - \int_\Omega \mathbf{u}_i(x) \cdot \nabla g(x) dx + \int_\Gamma g(x) \mathbf{u}_i(x) \cdot \mathbf{n} d\Gamma$$

For elements that have faces on the boundary:

```
CellTopology quad_4(shards::getCellTopologyData<Quadrilateral<4> >() );

Teuchos::RCP<Cubature<double> > quadCub = cubFactory.create(quad_4, cubDegree);
quadCub->getCubature(quadFacePoints, faceWeights);

CellTools::mapToReferenceSubcell(refFacePoints,
                                 quadFacePoints,
                                 2, iface, hex_8);

hexHCurlBasis.getValues(faceCVals, refFacePoints, OPERATOR_VALUE);

CellTools::setJacobian(faceJacobians,refFacePoints,
                       hexNodes, hex_8);

CellTools::setJacobianInv(faceJacobInv, faceJacobians );

fst::HCURLtransformVALUE<double>(hexCValsTrans,
                                 hexJacobInv,hexCVals);
```





$\hat{\mathbf{u}}(\hat{x}_{fp})$

$DF(\hat{x}_{fp})$

$DF^{-1}(\hat{x}_{fp})$

$DF^{-T}(\hat{x}_{fp})\hat{\mathbf{u}}_i(\hat{x}_{fp})$

Sandia National Laboratories

Step 3: Assemble right hand side

$$(\mathbf{b})_i = \int_\Omega \nabla \times \mathbf{u}_i(x) \cdot \mathbf{f}(x) dx - \int_\Omega \mathbf{u}_i(x) \cdot \nabla g(x) dx + \int_\Gamma g(x) \mathbf{u}_i(x) \cdot \mathbf{n} d\Gamma$$

```
CellTools::getPhysicalFaceNormals(hexFaceN,
                                  faceJacobians,
                                  iface, hex_8);
```

→ $\mathbf{n}_f$

← $DF(\hat{x}_{fp})$

Compute the dot product and multiply by weights

→ $\left(DF^{-T}\hat{\mathbf{u}}_i(\hat{x}_{fp})\right) \cdot \mathbf{n}_f J(\hat{x}_{fp}) \omega_{fp}$

```
fst::integrate<double>(rhsgBoundary,
                       gBoundaryData,
                       hexCValsDotNorm,
                       COMP_BLAS);
```

→ $\sum_{fp=1}^{numFPt} \left(DF^{-T}\hat{\mathbf{u}}_i(\hat{x}_{fp})\right) \cdot \mathbf{n}_f J(\hat{x}_{fp}) \omega_{fp} \left(g(\hat{x}_{fp})\right)$

← $g(\hat{x}_{fp})$

← $\left(DF^{-T}\hat{\mathbf{u}}_i(\hat{x}_{fp})\right) \cdot \mathbf{n}_f J(\hat{x}_{fp}) \omega_{fp}$

```
fst::applyFieldSigns<double>(rhsgBoundary,
                             hexBasisSigns);
```

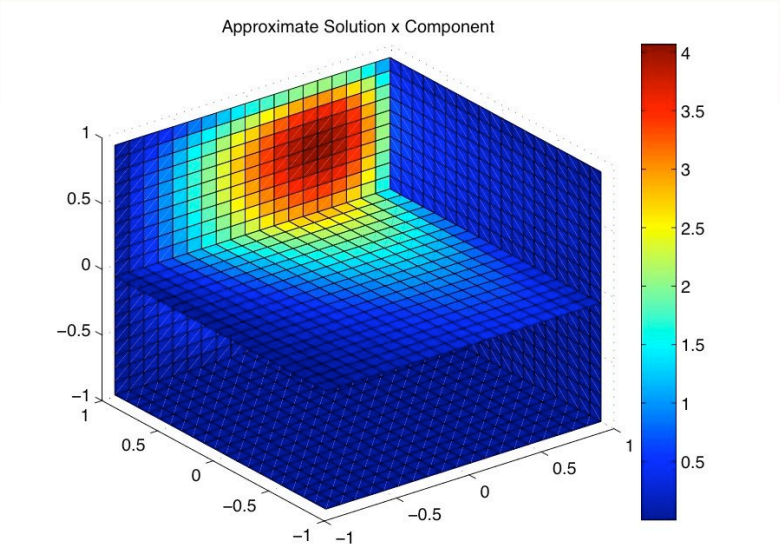
↔ $\sum_{fp=1}^{numFPt} \sigma_i \left(DF^{-T}\hat{\mathbf{u}}_i(\hat{x}_{fp})\right) \cdot \mathbf{n}_f J(\hat{x}_{fp}) \omega_{fp} \left(g(\hat{x}_{fp})\right)$

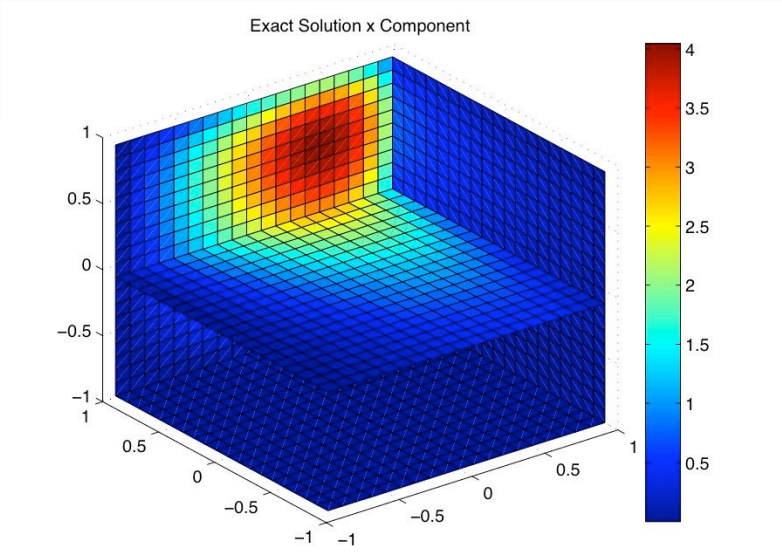
← σ_i

$$(\mathbf{b})_i \approx \sum_{p=1}^{numPt} \sigma_i \left(J^{-1} DF(\nabla \times \hat{\mathbf{u}}_i)(\hat{x}_p)\right) \cdot \left(\mathbf{f}(\hat{x}_p)\right) J(\hat{x}_p) \omega_p - \sum_{p=1}^{numPt} \sigma_i \left(DF^{-1}(\hat{\mathbf{u}}_i)(\hat{x}_p)\right) \cdot \left(\nabla g(\hat{x}_p)\right) J(\hat{x}_p) \omega_p$$

$$+ \sum_{f=1}^{nBndF} \sum_{fp=1}^{numFPt} \sigma_i \left(DF^{-T}\hat{\mathbf{u}}_i(\hat{x}_{fp})\right) \cdot \mathbf{n}_f \left(g(\hat{x}_{fp})\right) J(\hat{x}_{fp}) \omega_{fp}$$

Step 4: Solve

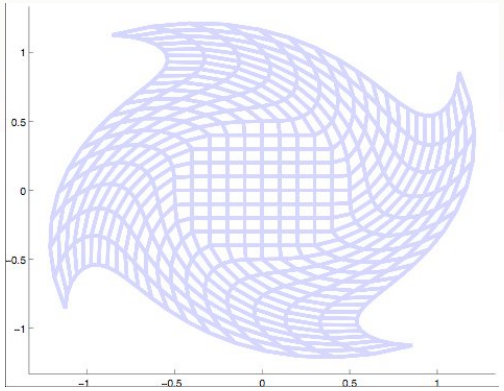




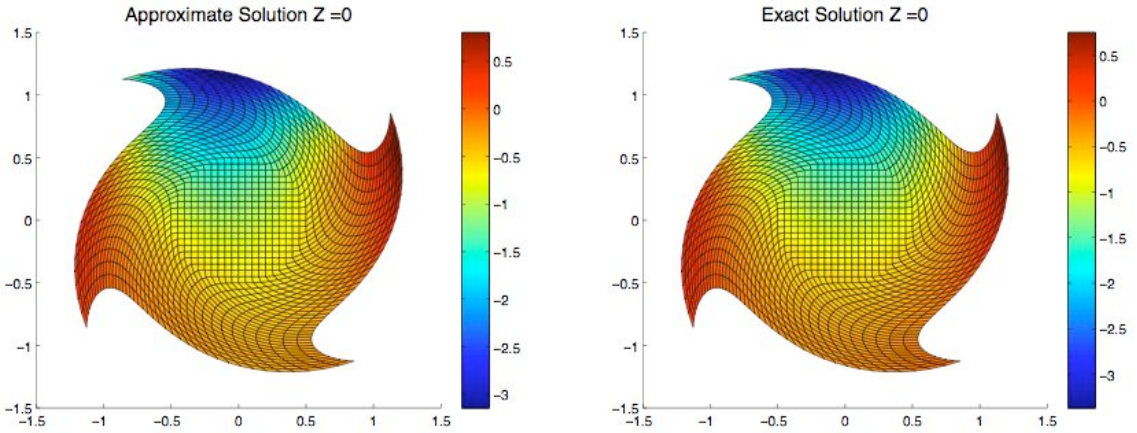
Error	Mesh size			Order
	10x10x10	20x20x20	40x40x40	
L2	0.31318	0.13991	0.067749	**1.1044**
H(curl)	2.3036	1.1575	0.57905	**0.9961**

Sandia National Laboratories

Step 4: Solve



2D (x-y) projection of the 3D model distorted mesh problem at the coarsest grid resolution.



3D Hex Mesh generated by PAMGEN (Trilinos package)

Error	Mesh size			Order
	10x10x4	20x20x8	40x40x16	
L2	0.89131	0.420695	0.212596	**0.98466**
H(curl)	4.89925	2.73187	1.55896	**0.80930**

L2 is optimal but H(curl) - suboptimal for non-affine Hex; see Falk et al. 2009

Sandia National Laboratories