Variational and Geometric Aspects of Compatible Discretizations
Part III

EU Regional School Part 2: Short Courses 11 and 12
October 1-2, 2009

Pavel Bochev
Applied Mathematics and Applications
Sandia National Laboratories

Supported in part by

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.
Overview

• **Part 1: Variational Compatibility**
  – Crash course in variational methods
  – Variational compatibility of FEM and what it means
  – Alternatives to variationally compatible FEM

• **Part 2: Geometric Compatibility**
  – What it is
  – What it buys you
  – How it relates to variational compatibility

• **Part 3: Software for compatible discretizations**
  – The evolving scope of the Trilinos project: the capability areas.
  – Intrepid: interoperable tools for compatible discretizations
  – Examples
Sandia Physics Simulation Codes

• **Element-based**
  – Finite element, finite volume, finite difference, network, etc…

• **Large-scale**
  – Billions of unknowns

• **Parallel**
  – MPI-based SPMD
  – Distributed memory

• **C++**
  – Object oriented
  – Some coupling to legacy Fortran libraries

- Fluids
- Combustion
- MEMS
- Circuits
- Structures
- Plasmas
- Circuits
Motivation For Trilinos

• Sandia does LOTS of solver work.
• 10 years ago …
  – Aztec was a mature package. Used in many codes.
  – FETI, PETSc, DSCPack, Spooles, ARPACK, DASPK, and many other codes were (and are) in use.
  – New projects were underway or planned in multi-level preconditioners, eigensolvers, non-linear solvers, etc…

• The challenges:
  – Little or no coordination was in place to:
    • Efficiently reuse existing solver technology.
    • Leverage new development across various projects.
    • Support solver software processes.
    • Provide consistent solver APIs for applications.
  – ASCI was forming software quality assurance/engineering (SQA/SQE) requirements:
    • Daunting requirements for any single solver effort to address alone.
Evolving Trilinos Solution: Package Concept

• **Trilinos**\(^1\) is an evolving framework to address these challenges:
  – Fundamental atomic unit is a *package*.
  – Includes core set of *vector*, *graph* and *matrix* classes (Epetra/Tpetra packages).
  – Provides a common abstract solver API (Thyra package).
  – Provides a ready-made package infrastructure (new_package package):
    • Source code management (cvs, bonsai).
    • Build tools: CMake.
    • Automated regression testing (queue directories within repository).
    • Communication tools (mailman mail lists).
  – **Specifies requirements and suggested practices for package SQA.**

• **In general allows us to categorize efforts:**
  – Efforts best done at the Trilinos level (useful to most or all packages).
  – Efforts best done at a package level (peculiar or important to a package).

⇒ **Allows package developers to focus only on things that are unique to their package.**

\(^1\) Trilinos loose translation: “A string of pearls”
Evolving Trilinos Solution

- Trilinos grew beyond a “solvers” framework
- Natural expansion of capabilities to satisfy application and research needs

\[ L(u) = f \]
Math. model

\[ L_h(u_h) = f_h \]
Numerical model

\[ u_h = L_h^{-1} f_h \]
Algorithms

**Numerical math**
Convert to models that can be solved on digital computers

**Algorithms**
Find faster and more efficient ways to solve numerical models

- Functionality types mapped into capability areas
- Interoperable tools for scientific computing
- Enables rapid development of production codes

Discretizations
- Time domain
- Space domain

Methods
- Automatic diff.
- Domain dec.
- Mortar methods

Solvers
- Linear
- Nonlinear
- Eigenvalues
- Optimization

Trilinos core
- Petra
- Utilities
- Interfaces
- Load Balancing

Computation

Physics
Evolving Trilinos Solution: Capability Areas

Current Capability areas of Trilinos:

1. Framework & Tools - Jim Willenbring
2. Software Engineering Technologies and Integration - Roscoe Bartlett
3. Discretizations - Pavel Bochev
4. Meshes, Geometry, & Load Balancing - Karen Devine
5. Scalable Linear Algebra - Michael Heroux
6. Linear & Eigen Solvers - Jonathan Hu
7. Embedded Nonlinear Analysis Tools - Andy Salinger

http://trilinos.sandia.gov/capability_areas.html
Discretization Capability Area

A collection of packages and interfaces for rapid development of application codes for the numerical solution of PDE and related problems: e.g., optimal control and design, inverse problems,… using Embedded Tools from Capability area 7.

3 related categories of discretization tools:

• **Global tools:** **FEI** (A. Williams)
  – Interfaces for assembling local (cell-wise) data into global representations of discrete differential operators and functionals by matrices and vectors

• **Model tools:** **PHALANX** (R. Pawlowski)
  – Manage discrete representations of physical fields in the PDE model and their relationships necessary to form Jacobians, residuals, operator matrices, etc.

• **Local tools:** **INTREPID, SHARDS** (P. Bochev, D. Ridzal, K. Peterson, C. Edwards)
  – Provide basic building blocks for numerical PDEs such as reconstruction operators (bases) for FE, FV and FD methods on a single cell, pullback (transformation of bases), cell and subcell integration.

## Trilinos Package Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Objective</th>
<th>Package(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Core</strong></td>
<td>Linear algebra objects</td>
<td>Epetra, Jpetra, Tpetra</td>
</tr>
<tr>
<td></td>
<td>Abstract interfaces</td>
<td>Thyra, Stratimikos, RTOp</td>
</tr>
<tr>
<td></td>
<td>Load Balancing</td>
<td>Zoltan, Isorropia</td>
</tr>
<tr>
<td></td>
<td>&quot;Skins&quot;</td>
<td>PyTrilinos, WebTrilinos, Star-P, <em>ForTrilinos</em></td>
</tr>
<tr>
<td></td>
<td>C++ utilities, (some) I/O</td>
<td>Teuchos, EpetraExt, Kokkos, Triutils</td>
</tr>
<tr>
<td><strong>Solvers</strong></td>
<td>Iterative (Krylov) linear solvers</td>
<td>AztecOO, Belos, Komplex</td>
</tr>
<tr>
<td></td>
<td>Direct linear solvers: sparse/dense</td>
<td>Amesos/Epetra, Teuchos, Pliris</td>
</tr>
<tr>
<td></td>
<td>Iterative eigenvalue solvers</td>
<td>Anasazi</td>
</tr>
<tr>
<td></td>
<td>Preconditioners : ILU-type/ML/Block</td>
<td>AztecOO, IFPACK/ ML, CLAPS/ Meros</td>
</tr>
<tr>
<td></td>
<td>Nonlinear system solvers</td>
<td>NOX, LOCA</td>
</tr>
<tr>
<td></td>
<td>Optimization (SAND)</td>
<td>MOOCHO, Aristos</td>
</tr>
<tr>
<td><strong>Discretizations</strong></td>
<td>Spatial Discretizations/Assembly/grid</td>
<td>Intrepid, FEI, PAMGEN</td>
</tr>
<tr>
<td></td>
<td>Cell topology &amp; Arrays</td>
<td>Shards</td>
</tr>
<tr>
<td></td>
<td>Physics kernels</td>
<td>Phalanx</td>
</tr>
<tr>
<td></td>
<td>Time Integration</td>
<td>Rythmos</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td>Automatic Differentiation</td>
<td>Sacado</td>
</tr>
<tr>
<td></td>
<td>Mortar Methods</td>
<td>Moertel</td>
</tr>
</tbody>
</table>

[http://trilinos.sandia.gov/packages/]
Trilinos Availability / Information

• Trilinos and related packages are available via LGPL.
• Current release (9.0) is “click release”. Unlimited availability.
  – 3100+ Downloads (not including internal Sandia users).
  – 3880 registered users:
    • 61% university, 11% industry, 15% gov’t.
    • 38% European, 33% US, 15% Asian.

• Trilinos Release 10: September 2008.

• Trilinos Awards:
  – 2004 R&D 100 Award.
  – SC2004 HPC Software Challenge Award.
  – Sandia Team Employee Recognition Award.
  – Lockheed-Martin Nova Award Nominee.

• More information:

• 7th Annual Trilinos User Group Meeting in November 2009 @ SNL
  – talks available for download
Trilinos Contributors

Chris Baker
Developer of Anasazi, RBGen, Tpetra

Ross Bartlett
Lead Developer of Thyra and Stratimikos
Developer of Rythmos

Pavel Bochev
Project Lead and Developer of Intrepid

Paul Boggs
Developer of Thyra

Eric Boman
Lead Developer of Isorropia
Developer of Zoltan

Todd Coffey
Lead Developer of Rythmos

David Day
Developer of Komplex and Intrepid

Karen Devine
Lead Developer of Zoltan

Clark Dohrmann
Developer of CLAPS

Michael Gee
Developer of ML, NOX

Bob Heaphy
Lead Developer of Trilinos SQA

Mike Heroux
Trilinos Project Leader
Lead Developer of Epetra, AztecOO, Kokkos, Komplex, IFPACK, Thyra, Tpetra
Developer of Amesos, Belos, EpetraExt, Jpetra

Ulrich Hetmaniuk
Developer of Anasazi

Robert Hoekstra
Lead Developer of EpetraExt
Developer of Epetra, Thyra, Tpetra

Russell Hooper
Developer of NOX

Vicki Howle
Lead Developer of Meros
Developer of Belos and Thyra

Jonathan Hu
Developer of ML

Sarah Knepper
Developer of Komplex

Tammy Kolda
Lead Developer of NOX

Joe Kotulski
Lead Developer of Pliris

Rich Lehoucq
Developer of Anasazi and Belos

Kevin Long
Lead Developer of Thyra,
Developer of Teuchos

Roger Pawlowski
Lead Developer of NOX

Eric Phipps
Developer of LOCA, NOX, and Sacado

Denis Ridzal
Lead Developer of Aristos and Intrepid

Marzio Sala
Lead Developer of Didasko and IFPACK
Developer of ML, Amesos

Andrew Salinger
Lead Developer of LOCA

Paul Sexton
Developer of Epetra and Tpetra

Bill Spotz
Lead Developer of PyTrilinos
Developer of Epetra, New_Package

Ken Stanley
Lead Developer of Amesos and New_Package

Heidi Thornquist
Lead Developer of Anasazi, Belos, RBGen, and Teuchos

Ray Tuminaro
Lead Developer of ML and Meros

Jim Willenbring
Developer of Epetra and New_Package.
Trilinos library manager

Alan Williams
Lead Developer of Isorropia
Developer of Epetra, EpetraExt, AztecOO, Tpetra
Motivation For **Intrepid** (looks familiar…) 

- Sandia does LOTS of numerical PDEs work.  
- 10 years ago …  
  - MPSalsa was first-of-its kind **massively parallel FEM code** for chemically reacting flows: stabilized Q1-Q1 formulation (hardwired), spurred the Aztec solver  
  - Many other code development efforts: FEM: **Alegra** (shock-hydro), FV: **Premo**,…  
  - New projects for **FEM/FV/FD codes** were underway or planned in compressible flows, solids, fluids, combustion, e.g., **Sierra** framework  

- **The challenges:**  
  - **Improve coordination to:**  
    - Efficiently reuse existing discretization technology.  
    - Leverage new development across various projects.  
    - Support discretization software processes.  
    - Provide **consistent discretization APIs** for applications using FEM, FV or FD.  
  - **Meeting ASC software quality assurance/engineering (SQA/SQE) requirements**
Intrepid

INteroperable Tools for Rapid dEveloPment of compatIble Discretizations

A Trilinos package for compatible discretizations:

- Limited release as Trilinos 9.0 package in 2008
- Full release with Trilinos 10.0 in 2009 of the “expert” FEM interface
- Up to order 10 $H(\text{grad})$, $H(\text{curl})$ and $H(\text{div})$ FE bases on Quad, Triangle, Tetrahedron, Hexahedron, and Wedge cell topologies
- High quality cubature, e.g., positive weights only on Tri and Tet cells

When fully deployed (~2011) will provide

- support for hybrid discretizations (FEM, FV and FD)
- support for FV and FD on polygons and polyhedra
- optimized multi-core kernels
- optimized assembly (R. Kirby)
- Access to FEM, FV and FD methods using a common “user friendly” API

Developers: P. Bochev, D. Ridzal, K. Peterson, R. Kirby

http://trilinos.sandia.gov/packages/intrepid/
Why Focus on Compatible Discretizations?

Because they mitigate information loss inherent to discretization!

Recall: all discrete structures in our mimetic framework are induced by 2 basic operations
Mimetic framework guides the software design for compatible discretizations:

- common API provides simultaneous access to FEM, FV and FD methods
- supports hybrid discretizations (FEM, FV and FD) on unstructured grids

FEM, FV and FD methods are defined by choosing a specific reconstruction operator $I$:

**Direct:** $I$ is low order, easily extendable to arbitrary cells

**Pullback:** $I$ is defined on standard cells, easily extendable to high orders
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 neither one of the above categories
Mapping Functionality to Software

Cell Topology
- Cell Geometry
- Cell Integration
  - Discrete Spaces
  - Discrete Operators
  - Discrete Functionals
- Utilities

Cell Tools
  - Integration
    - Basis
      - FunctionSpaceTools
  - RealSpaceTools
    - ArrayTools
      - FieldContainer
        - Polylib

src/Cell
src/Discretization
src/Shared
The Basis Nomenclature

name = Space + Cell Name + Discrete Space + Order + Basis Type

- **HGRAD**
- **TRI**
- **C(omplete)**

- **HCURL**
- **TET**
- **I(ncomplete)**

- **HDIV**
- **QUAD**
- **B(roken)**

- **HVOL**
- **HEX**

- **HGRADVEC**
- **WEDGE**

- **HGRADTEN**
- **PYRAMID**

Example

- **HGRAD_TRI_C1_FEM** Linear (C1) Lagrange elements on Triangles
- **HGRAD_QUAD_C1_FEM** Bi-linear (C1) Lagrange elements on a Quad
- **HCURL_HEX_I1_FEM** Lowest-order Nedelec element of the 1st kind (I1) on a Hex
- **HCURL_HEX_C1_FEM** Lowest-order Nedelec element of the 2nd kind (C1) on a Hex
Basic Principles

Uses simple data structures!

- Multi-indexed Scalar value is the “only” data type in Intrepid. Implemented as multi-dimensional array (MDA): contiguous data layout with multi-index access.
- Intrepid methods are templated on Scalar and one or more Array types
- Users can retain their data arrays as long as they provide minimal interface
  - `My_array.rank()`
  - `My_array.dimension(dim_i)`
  - `My_array.size()`
  - `My_array(i),..., My_array(i,j,k,l,m)`
  - `My_array[i]`

- Intrepid provides its own implementation of a runtime, lexicographical multi-dimensional array FieldContainer:
  - similar to `std::vector` but with multi-index access and “view” option
  - Can be resized at runtime
  - Can be re-shaped at runtime

- Shards provides compile time MD Arrays using template meta-programming, for both lexicographical and “FORTRAN” style arrays.
Multi-dimensional Array (MDA) Conventions

I/O arguments in Intrepid are typically of the following two kinds

- **Data arrays:** multi-indexed Scalar without a Field index
- **Field arrays:** multi-indexed Scalar with a Field index

MDA index notation used in Intrepid:

<table>
<thead>
<tr>
<th>Index</th>
<th>Refers to</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>cell</td>
<td>[0, C)</td>
</tr>
<tr>
<td>n,v</td>
<td>node and vertex</td>
<td>[0, N), [0, V)</td>
</tr>
<tr>
<td>f</td>
<td>field</td>
<td>[0, F)</td>
</tr>
<tr>
<td>l</td>
<td>left field</td>
<td>[0, L)</td>
</tr>
<tr>
<td>r</td>
<td>right field</td>
<td>[0, R)</td>
</tr>
<tr>
<td>i,j,d1,d2...</td>
<td>spatial coordinate</td>
<td>[0, D)</td>
</tr>
<tr>
<td>p</td>
<td>point</td>
<td>[0, P)</td>
</tr>
<tr>
<td>k</td>
<td>multiset (Dk operators)</td>
<td>[0,</td>
</tr>
</tbody>
</table>

Multi-index examples:

- (c, p)
- (c, p, f)
- (c, p, r, i)
- (p, f)
- (p, l, i)
- (p, f, i, j)
- (c, f, p, k)
### Multi-dimensional Arrays (cont’d)

#### Field arrays

<table>
<thead>
<tr>
<th>Multi-index field</th>
<th>represents</th>
<th>container</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_f, u_l, u_r$</td>
<td>Scalar field</td>
<td>$(C, F, P)$</td>
</tr>
<tr>
<td>$u^i_f, u^i_l, u^i_r$</td>
<td>Vector field</td>
<td>$(C, F, P, D)$</td>
</tr>
<tr>
<td>$u^{ij}_f, u^{ij}_l, u^{ij}_r$</td>
<td>Tensor field</td>
<td>$(C, F, P, D, D)$</td>
</tr>
</tbody>
</table>

#### Data arrays

<table>
<thead>
<tr>
<th>Multi-index data</th>
<th>represents</th>
<th>container</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\kappa \ diag(\kappa, \ldots, \kappa)$</td>
<td>Scalar data</td>
<td>$(C, P)$</td>
</tr>
<tr>
<td>$a_i, b_i$</td>
<td>Vector data</td>
<td>$(C, P, D)$</td>
</tr>
<tr>
<td>$K_i$</td>
<td>Tensor data (diagonal)</td>
<td>$(C, P, D)$</td>
</tr>
<tr>
<td>$K_{ij}$</td>
<td>Tensor data (full)</td>
<td>$(C, P, D, D)$</td>
</tr>
</tbody>
</table>
Some Examples

Array types returned by `getValues` method of the `Basis` class

<table>
<thead>
<tr>
<th>operator/field rank</th>
<th>rank 0</th>
<th>rank 1 2D/3D</th>
<th>rank 2 2D/3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE</td>
<td>(F,P)</td>
<td>(F,P,D)</td>
<td>(F,P,D,D)</td>
</tr>
<tr>
<td>GRAD, D1</td>
<td>(F,P,D)</td>
<td>(F,P,D,D)</td>
<td>(F,P,D,D,D)</td>
</tr>
<tr>
<td>CURL</td>
<td>(F,P,D)(undef. in 3D)</td>
<td>(F,P)/(F,P,D)</td>
<td>(F,P,D)/(F,P,D,D)</td>
</tr>
<tr>
<td>DIV</td>
<td>(F,P,D)(only in 1D)</td>
<td>(F,P)</td>
<td>(F,P,D)</td>
</tr>
<tr>
<td>D1,D2,...,D10</td>
<td>(F,P,K)</td>
<td>(F,P,D,K)</td>
<td>(F,P,D,D,K)</td>
</tr>
</tbody>
</table>

Jacobian computation by `CellTools` methods:

```cpp
template<class ArrayScalar>
static void setJacobian(ArrayScalar& jacobian,
                         const ArrayScalar& points,
                         const ArrayScalar& nodes,
                         const shards::CellTopology& cellTopo,
                         const int& whichCell = -1);
```
Working With all Types of Finite Element Spaces is Very Easy!

- Implementation follows the canonical Ciarlet definition:
  - Finite element $= \{\kappa, P, \Lambda\}$
    - $\kappa$ - element or cell
    - $P$ - discrete space, not necessarily polynomial
    - $\Lambda$ - set of unisolvent DoF (linear functionals $P \rightarrow R$)
  - $\{\kappa, P, \Lambda\}$ is defined by pullback from $\{\hat{\kappa}, \hat{P}, \hat{\Lambda}\}$
    - $\Phi^*: \hat{P}(\hat{\kappa}) \rightarrow P(\kappa)$

$\Rightarrow$ Basis is defined on a reference cell and is dual to a DoF set
$\Rightarrow$ Basis on a physical cell is defined by pullback
$\Rightarrow$ Require cell topology with a reference cell
Reference Cell Basis Example

Every basis function and its dual DoF functional are assigned a

- Unique DoF ordinal
- Unique 4-field DoF tag that associates the DoF with a subcell

<table>
<thead>
<tr>
<th>DoF ordinal</th>
<th>subc dim</th>
<th>subc ordinal</th>
<th>subc DoF ord</th>
<th>subc num DoF</th>
<th>DoF definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$L_0(u) = (u.t)(0.5,0,0)$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$L_1(u) = (u.t)(0.5,0.5,0)$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>$L_2(u) = (u.t)(0,0.5,0)$</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>$L_3(u) = (u.t)(0,0,0.5)$</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>$L_4(u) = (u.t)(0.5,0,0.5)$</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>$L_5(u) = (u.t)(0,0.5,0.5)$</td>
</tr>
<tr>
<td>MAX</td>
<td>maxScDim=1</td>
<td>maxScOrd=5</td>
<td>maxDfOrd=0</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

What kind of basis is this?

template<class Scalar, class ArrayScalar>
class Basis_HCURL_TET_I1_FEM : public Basis<Scalar, ArrayScalar>
**Pullback Definitions**

\[ \Phi_G^* : G(\hat{k}) \rightarrow G(\kappa) \quad \Phi_G^* (\hat{u}) = \hat{u} \circ F^{-1}_\kappa \]

\[ \Phi_C^* : C(\hat{k}) \rightarrow C(\kappa) \quad \Phi_C^* (\hat{u}) = \left( DF^{-T}_\kappa \cdot \hat{u} \right) \circ F^{-1}_\kappa \]

\[ \Phi_D^* : D(\hat{k}) \rightarrow D(\kappa) \quad \Phi_D^* (\hat{u}) = \left( J^{-1}_\kappa DF \cdot \hat{u} \right) \circ F^{-1}_\kappa \]

\[ \Phi_V^* : V(\hat{k}) \rightarrow V(\kappa) \quad \Phi_V^* (\hat{u}) = \left( J^{-1}_\kappa \hat{u} \right) \circ F^{-1}_\kappa \]

**Grad**

**Curl**

**Div**

**Volume**

Why do I care? “Change of variables” that **commutes** with grad, curl and div

\[
\nabla \Phi_G^* (\hat{u}) = \Phi_C^* (\nabla \hat{u}) = \left( DF^{-T}_\kappa \cdot (\nabla \hat{u}) \right) \circ F^{-1}_\kappa
\]

\[
\nabla \times \Phi_C^* (\hat{u}) = \Phi_D^* (\nabla \times \hat{u}) = \left( J^{-1}_\kappa DF \cdot (\nabla \times \hat{u}) \right) \circ F^{-1}_\kappa
\]

\[
\nabla \cdot \Phi_D^* (\hat{u}) = \Phi_V^* (\nabla \cdot \hat{u}) = \left( J^{-1}_\kappa (\nabla \cdot \hat{u}) \right) \circ F^{-1}_\kappa
\]

Where do I find the pieces to put together finite element spaces?

\[ \hat{u}(\hat{x}_p), \hat{d}(\hat{x}_p), \hat{p}(\hat{x}_p) \ldots \rightarrow \text{Basis} \quad \text{basis values at ref. cell} \]

\[ DF(\hat{x}_p), J(\hat{x}_p) \rightarrow \text{CellTools} \quad \text{pullback ingredients} \]

\[ u(F^{-1}_\kappa(\hat{x}_p)), du(F^{-1}_\kappa(\hat{x}_p)) \rightarrow \text{FunctionSpaceTools} \quad \text{application of pullback} \]
Intrepid Example: Compatible LSFEM for div-curl Systems

Magnetostatics is a div-curl system

\[
\begin{align*}
\nabla \times \mathbf{H} &= \mathbf{J} \quad \text{in } \Omega \\
\nabla \cdot \mu \mathbf{H} &= 0 \quad \text{in } \Omega \\
\mathbf{n} \times \mathbf{H} &= 0 \quad \text{on } \partial \Omega
\end{align*}
\]

\( \mu \) magnetic permeability

\[
\begin{align*}
\nabla \times \mathbf{B} &= \mathbf{J} \quad \text{in } \Omega \\
\nabla \cdot \mathbf{B} &= 0 \quad \text{in } \Omega \\
\mathbf{n} \cdot \mathbf{B} &= 0 \quad \text{on } \partial \Omega
\end{align*}
\]

Diffusion is a div-curl system

\[
\begin{align*}
\nabla \times \mathbf{u} &= 0 \quad \text{in } \Omega \\
\nabla \cdot \mathbf{u} &= g \quad \text{in } \Omega \\
\mathbf{n} \times \mathbf{u} &= 0 \quad \text{on } \partial \Omega
\end{align*}
\]

\( \nabla \times \mathbf{u} = 0 \Rightarrow \mathbf{u} = -\nabla p \quad \text{in } \Omega \)

\( \nabla \cdot \mathbf{u} = 0 \Rightarrow p = 0 \quad \text{on } \partial \Omega \)

The incompressible Stokes equations are 2 coupled div-curl systems:

\[
\begin{align*}
\nabla \times \omega + \nabla p &= 0 \quad \text{in } \Omega \\
\nabla \cdot \omega &= 0 \quad \text{in } \Omega \\
\mathbf{n} \times \omega &= 0 \quad \text{on } \partial \Omega
\end{align*}
\]

\[
\begin{align*}
\nabla \times \mathbf{u} &= \omega \quad \text{in } \Omega \\
\nabla \cdot \mathbf{u} &= 0 \quad \text{in } \Omega \\
\mathbf{n} \cdot \mathbf{u} &= 0 \quad \text{on } \partial \Omega
\end{align*}
\]

\( \omega \) vorticity

\( \mathbf{u} \) velocity

\( p \) pressure

**Div-curl systems are everywhere**, but direct Galerkin methods lead to linear systems that are hard to solve by iterative methods. Better systems can be obtained by LSFEM. This (nontrivial) example builds an LSFEM code for div-curl systems with tangential B.C.
A Least-Squares Principle

A model Div-Curl System

\[
\begin{align*}
\nabla \times \mathbf{u} &= \mathbf{f} \quad \text{in } \Omega \\
\nabla \cdot \mu \mathbf{u} &= g \quad \text{in } \Omega \\
n \times \mathbf{u} &= 0 \quad \text{on } \partial \Omega
\end{align*}
\]

\(\Omega \subset \mathbb{R}^3\) → bounded contractible domain

\(\partial \Omega\) → Lipschitz continuous

A continuous least-squares principle

\[
\begin{align*}
J(\mathbf{u};\mathbf{f},g) &= \|\nabla \times \mathbf{u} - \mathbf{f}\|_{0,\Omega_2}^2 + \|\nabla \cdot \mathbf{u} - g\|_{0,\Omega_3}^2 \\
X &= H_0(\text{curl},\Omega) \cap H(\text{div},\Omega)
\end{align*}
\]

\[
\min_{\mathbf{u}} J(\mathbf{u};\mathbf{f},g)
\]

Norm-equivalence (Friedrichs Inequality)

\[
\|\mathbf{u}\|_{DC} \leq C \left( \|\nabla \times \mathbf{u}\|_{0,\Omega_2} + \|\nabla \cdot \mathbf{u}\|_{0,\Omega_3} \right) \quad \forall \mathbf{u} \in H_0(\text{curl},\Omega) \cap H(\text{div},\Omega)
\]

Lax-Milgram + norm-equivalence → coercivity → unique least-squares solution.

Moreover, that solution coincides with the solution of the original div-curl system.
Why not Conforming LSFEM?

A straightforward **conforming** discrete least-squares principle

\[
J(u_h; f, g) = \left\| \nabla \times u_h - f \right\|^2_{0,\Omega} + \left\| \nabla \cdot u_h - g \right\|^2_{0,\Omega}
\]

\[
X_h \subset H_0(\text{curl}, \Omega) \cap H(\text{div}, \Omega)
\]

The trouble with this LSFEM

\[
\begin{align*}
X_h \subset H_0(\text{curl}, \Omega) & \Rightarrow \text{Tangential continuity} \\
\Rightarrow \quad C^0 \text{ continuity} & \Rightarrow X_h \subset H^1(\Omega) \cap H_0(\text{curl}, \Omega)
\end{align*}
\]

\[
X_h \subset H(\text{div}, \Omega) \Rightarrow \text{Normal continuity}
\]

Why is this bad?

Costabel (1991) shows that unless \( \Omega \) has smooth boundary or is a convex polyhedron, \( H^1 \cap H_0(\text{curl}) \) may have

infinite co-dimension in \( H_0(\text{curl}) \cap H(\text{div}) \)

\( \Rightarrow \) a \( C^0 \) (nodal) finite element space may lose approximability property in \( H_0(\text{curl}) \cap H(\text{div}) \), i.e., solution will not converge.

\( \Rightarrow \) Mixed methods can solve this, but we have another approach…
“Semi-Conforming” LSFEM

A remedy: Give up on being div-conforming and define “semi-conforming” LSFEM

\[
\begin{aligned}
J(u_h; f, g) &= \| \nabla \times u_h - f \|_{0, \Omega}^2 + \| \nabla^* \cdot u_h - g \|_{0, \Omega}^2 \\
X_h &\subset H_0(curl, \Omega) \cap H(div, \Omega)
\end{aligned}
\]

\(X_h = C^h_0(\Theta_1)\) 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!

Everybody knows how to define a weak divergence:

\[
\nabla^* \cdot C^h_0(\Theta_1) \rightarrow G_0^h(\Theta_0); \quad p_h = \nabla^* \cdot u_h \iff (p_h, q_h)_{0, \Theta_0} = -(u_h, \nabla q_h)_{0, \Theta_i} \quad \forall q_h \in G_0^h
\]

Norm-equivalence of the semi-conforming LSFEM (Discrete Friedrichs Inequality)

\[
\| u_h \|_{D^*C} \leq C \left( \| \nabla \times u_h \|_{0, \Theta_2} + \| \nabla^* \cdot u_h \|_{0, \Theta_0} \right) \quad \forall u_h \in C_0^h \quad \text{(effect of compatibility)}
\]

Error estimates

\[
\| \nabla \times (u - u_h) \|_0 + \| \nabla \cdot u_h - \nabla^* \cdot u_h \|_0 \leq Ch' \left( \| \nabla \times u \|_r + \| \nabla \cdot u \|_r \right)
\]
From Discretization to Linear System

For clarity we now focus on the lowest-order case

\[ C_h^0(\Theta_1) \rightarrow \text{the lowest-order Nedelec space of the 1st kind} \]

\[ C_h^0(\Theta_0) \rightarrow \text{the lowest-order nodal } C^0 \text{ space} \]

Discrete least-squares problem

\[
(\nabla \times u_h, \nabla \times v_h)_{0,\Theta_2} + (\nabla_h^* \cdot u_h, \nabla_h^* \cdot v_h)_{0,\Theta_0} = (f, \nabla \times v_h)_{0,\Theta_2} + (g, \nabla_h^* \cdot v_h)_{0,\Theta_0} \quad \forall v_h \in C_h^0
\]

\[
\downarrow
\]

\[
K(\Theta_2) + M_C(\Theta_1)D_{VE}M^{-1}_G(\Theta_0)D^T_{VE}M_C(\Theta_1) = b \quad \text{curl-curl + grad-div matrices}
\]

where

\[
\begin{cases}
M_C(\Theta_1) & \text{curl-conforming mass matrix} \\
M_G(\Theta_0) & \text{grad-conforming mass matrix} \\
D_{VE} & \text{vertex-to-edge incidence matrix}
\end{cases}
\]

and

\[ \Theta_0 = \mu^{-1}; \quad \Theta_1 = 1; \quad \Theta_0 = \mu \]

- 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: \( K(\Theta_2), M_C(\Theta_1), M_G(\Theta_0), \text{ and } D_{VE} \)
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

5. **Have a beer or two**
Step 1: **define cell, cubature, and basis**

Define Cell Topology with Shards

```cpp
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

```cpp
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);
```
Step 1: **define cell, cubature, and basis**

Define Basis on reference cell

```cpp
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

```cpp
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

```cpp
int numCells = 100;
FieldContainer<double> hexWorkset(numCells, numNodesPerCell, spaceDim);
```

// Intialize hexWorkset from mesh data

Physical coordinates of cell nodes

Compute Jacobians of cell workset

```cpp
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);
```

$F : \hat{K} \rightarrow K$
Step 2: pullback data: $H(\text{curl})$ basis signs

Local (reference) cell

Global (physical) cell

FieldContainer<double> hexBasisSigns(numCells, numFieldsC);

$\hat{u}_0$, $\hat{u}_1$, $\hat{u}_2$, $\hat{u}_3$
Step 3: Assembly

**H(gradient) mass matrix**

\[
(M_G)_{ij} = \int_\Omega \phi_i(x) \phi_j(x) dx = \int_\Omega (\Phi^\ast \hat{\phi}_i)(x)(\Phi^\ast \hat{\phi}_j)(x) dx = \int_\Omega \hat{\phi}_i(\hat{x}) \hat{\phi}_j(\hat{x}) J(\hat{x}) d\hat{x} \\
\approx \sum_{p=1}^{\text{numPt}} \hat{\phi}_i(\hat{x}_p) \hat{\phi}_j(\hat{x}_p) J(\hat{x}_p) \omega_p
\]

**H(curl) mass matrix**

\[
(M_C)_{ij} = \int_\Omega \mathbf{u}_i(x) \cdot \mathbf{u}_j(x) dx = \int_\Omega (\Phi^\ast \hat{\mathbf{u}}_i)(x) \cdot (\Phi^\ast \hat{\mathbf{u}}_j)(x) dx \\
= \int_\Omega \sigma_i \sigma_j \left( \mathbf{D} \mathbf{F}^{-T} \hat{\mathbf{u}}_i(\hat{x}) \right) \cdot \left( \mathbf{D} \mathbf{F}^{-T} \hat{\mathbf{u}}_j(\hat{x}) \right) J(\hat{x}) d\hat{x} \\
\approx \sum_{p=1}^{\text{numPt}} \sigma_i \sigma_j \left( \mathbf{D} \mathbf{F}^{-T} \hat{\mathbf{u}}_i(\hat{x}_p) \right) \cdot \left( \mathbf{D} \mathbf{F}^{-T} \hat{\mathbf{u}}_j(\hat{x}_p) \right) J(\hat{x}_p) \omega_p
\]

**H(curl) stiffness matrix**

\[
(K_C)_{ij} = \int_\Omega \nabla \times \mathbf{u}_i(x) \cdot \nabla \times \mathbf{u}_j(x) dx = \int_\Omega \left( \Phi^\ast (\hat{\nabla} \times \hat{\mathbf{u}}_i) \right)(x) \cdot \left( \Phi^\ast (\hat{\nabla} \times \hat{\mathbf{u}}_j) \right)(x) dx \\
= \int_\Omega \sigma_i \sigma_j \left( \mathbf{D}^{-1} \mathbf{F} (\hat{\nabla} \times \hat{\mathbf{u}}_i)(\hat{x}) \right) \cdot \left( \mathbf{D}^{-1} \mathbf{F} (\hat{\nabla} \times \hat{\mathbf{u}}_j)(\hat{x}) \right) J(\hat{x}) d\hat{x} \\
\approx \sum_{p=1}^{\text{numPt}} \sigma_i \sigma_j \left( \mathbf{D}^{-1} \mathbf{F} (\hat{\nabla} \times \hat{\mathbf{u}}_i)(\hat{x}_p) \right) \cdot \left( \mathbf{D}^{-1} \mathbf{F} (\hat{\nabla} \times \hat{\mathbf{u}}_j)(\hat{x}_p) \right) J(\hat{x}_p) \omega_p
\]
Generic Intrepid Assembly Procedure (symmetric operators)

\[
(A_x)_{ij} = \int_{\Omega} L u_i(x) L u_j(x) \, dx = \int_{\Omega} \sigma_i \sigma_j (\Phi^* \hat{L} u_i(x)(\Phi^* \hat{L} u_j(x)) \, dx = \int_{\Omega} \sigma_i \sigma_j (T(DF) \hat{L} u_i(\hat{x}))(T(DF) \hat{L} u_j(\hat{x})) J(\hat{x}) \, d\hat{x}
\]

\[
\approx \sum_{p=1}^{numPt} \sigma_i \sigma_j (T(DF(\hat{x}_p)) \hat{L} u_i(\hat{x}_p))(T(DF(\hat{x}_p)) \hat{L} u_j(\hat{x}_p)) J(\hat{x}_p) \omega_p
\]

Basis->getValue

FST::applyTransform

FST::multiplyMeasure

FST::integrate

FST::applyL/RFieldSigns

\[
\sum_{p=1}^{numPt} \left( T(DF(\hat{x}_p)) \hat{L} u_i(\hat{x}_p) \right) \left( T(DF(\hat{x}_p)) \hat{L} u_j(\hat{x}_p) \right) J(\hat{x}_p) \omega_p
\]

\[
\sum_{p=1}^{numPt} \sigma_i \sigma_j (T(DF(\hat{x}_p)) \hat{L} u_i(\hat{x}_p))(T(DF(\hat{x}_p)) \hat{L} u_j(\hat{x}_p)) J(\hat{x}_p) \omega_p
\]
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> hexGVarsTrans(numCells, numFieldsG, numCubPoints);
FieldContainer<double> hexGVarsTransWeighted(numCells, numFieldsG, numCubPoints);

// Containers for element HCURL mass matrix
FieldContainer<double> massMatrixC(numCells, numFieldsC, numFieldsC);
FieldContainer<double> hexCVarsTrans(numCells, numFieldsC, numCubPoints, spaceDim);
FieldContainer<double> hexCVarsTransWeighted(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_FEChsMatrix MassG(Copy, globalMapG, numFieldsG);
Epetra_FEChsMatrix MassC(Copy, globalMapC, numFieldsC);
Epetra_FEChsMatrix StiffC(Copy, globalMapC, numFieldsC);
Epetra_FEChsVector rhsC(globalMapC);
Step 3: **Assemble H(\text{grad}) mass matrix**

\[
(M_G)_{ij} \approx \sum_{p=1}^{numPt} \hat{\phi}_i(\hat{x}_p) \hat{\phi}_j(\hat{x}_p) J(\hat{x}_p) \omega_p
\]
Step 3: assemble $H(\text{curl})$ mass matrix

\[
(M_C)_{ij} \approx \sum_{p=1}^{\text{numPt}} \sigma_i \sigma_j (DF^{-T}\hat{\mathbf{u}}_i(\hat{x}_p)) \cdot (DF^{-T}\hat{\mathbf{u}}_j(\hat{x}_p)) J(\hat{x}_p) \omega_p
\]
Step 3: assemble H(curl) stiffness matrix

\[
\begin{align*}
\text{fst} & : \text{HURL\_transform\_CURL\langle double\rangle} (\text{hexCurlsTrans}, \\
& \quad \text{hexJacob}, \\
& \quad \text{hexJacobianDet}, \\
& \quad \text{hexCVCs}) ; \\
& \quad J^{-1}(\hat{x}_p)DF(\hat{x}_p)(\hat{\nabla} \times \hat{u}_j)(\hat{x}_p) \\
& \quad \text{df} = DF(\hat{x}_p) \\
& \quad \text{J} = J(\hat{x}_p) \\
& \quad (\hat{\nabla} \times \hat{u}_j)(\hat{x}_p) \\
\text{fst} & : \text{multiply\_measure\langle double\rangle} (\text{hexCurlsTransWght}, \\
& \quad \text{reuse} \rightarrow \text{cell\_measure}, \\
& \quad \text{hexCurlsTrans}) ; \\
& \quad J^{-1}(\hat{x}_p)DF(\hat{x}_p)(\hat{\nabla} \times \hat{u}_j)(\hat{x}_p)(J(\hat{x}_p)\omega_p) \\
& \quad \text{J} = J(\hat{x}_p)\omega_p \\
& \quad J^{-1}(\hat{x}_p)DF(\hat{x}_p)(\hat{\nabla} \times \hat{u}_j)(\hat{x}_p) \\
& \quad J^{-1}(\hat{x}_p)DF(\hat{x}_p)(\hat{\nabla} \times \hat{u}_j)(\hat{x}_p)(J(\hat{x}_p)\omega_p) \\
\text{fst} & : \text{integrate\langle double\rangle} (\text{stiff\_matrix\_C}, \\
& \quad \text{hexCurlsTrans}, \\
& \quad \text{hexCurlsTransWght}, \\
& \quad \text{COMP\_BLAS}) ; \\
& \quad \sum_{p=1}^{\text{numPt}} \left( J^{-1}(\hat{x}_p)DF(\hat{\nabla} \times \hat{u}_j)(\hat{x}_p) \right) \cdot \left( J^{-1}(\hat{x}_p)DF(\hat{\nabla} \times \hat{u}_j)(\hat{x}_p) \right) J(\hat{x}_p)\omega_p \\
\text{fst} & : \text{apply\_left\_field\_signs\langle double\rangle} (\text{stiff\_matrix\_C}, \\
& \quad \text{hexBasis\_signs}) ; \\
& \quad \sum_{p=1}^{\text{numPt}} \sigma_i \left( J^{-1}(\hat{x}_p)DF(\hat{\nabla} \times \hat{u}_j)(\hat{x}_p) \right) \cdot \left( J^{-1}(\hat{x}_p)DF(\hat{\nabla} \times \hat{u}_j)(\hat{x}_p) \right) J(\hat{x}_p)\omega_p \quad \text{for } \sigma_i \\
\text{fst} & : \text{apply\_right\_field\_signs\langle double\rangle} (\text{stiff\_matrix\_C}, \\
& \quad \text{hexBasis\_signs}) ; \\
& \quad \sum_{p=1}^{\text{numPt}} \sigma_j \left( J^{-1}(\hat{x}_p)DF(\hat{\nabla} \times \hat{u}_j)(\hat{x}_p) \right) \cdot \left( J^{-1}(\hat{x}_p)DF(\hat{\nabla} \times \hat{u}_j)(\hat{x}_p) \right) J(\hat{x}_p)\omega_p \quad \text{for } \sigma_j \\
\end{align*}
\]

\[
K_{ij} = \sum_{p=1}^{\text{numPt}} \sigma_i \sigma_j \left( J^{-1}(\hat{x}_p)DF(\hat{\nabla} \times \hat{u}_j)(\hat{x}_p) \right) \cdot \left( J^{-1}(\hat{x}_p)DF(\hat{\nabla} \times \hat{u}_j)(\hat{x}_p) \right) J(\hat{x}_p)\omega_p
\]
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)

<table>
<thead>
<tr>
<th>Error</th>
<th>Mesh size</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10x10x4</td>
<td>20x20x8</td>
</tr>
<tr>
<td>L2</td>
<td>0.89131</td>
<td>0.420695</td>
</tr>
<tr>
<td>H(curl)</td>
<td>4.89925</td>
<td>2.73187</td>
</tr>
</tbody>
</table>

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

Further examples:
- Trilinos/packages/trilinoscouplings/examples/scaling
- Trilinos/packages/intrepid/example/Drivers
Summary

- Trilinos is rapidly evolving into a powerful tool for parallel scientific computing

- Trilinos capabilities are organized in 7 areas::
  - Frameworks and tool
  - Software engineering technologies and integration
  - Discretizations
  - Meshes, geometry and load balancing
  - Scalable linear algebra
  - Linear and eigsolvers
  - Embedded nonlinear analysis tools

- Intrepid is part of Trilinos and belongs in the discretization capability area
  - A suite of tools for compatible discretizations
  - Can be used in your code as an “element” library
  - Can be used to develop new finite element codes
  - Uses simple, yet powerful and efficient data structures (MD Array)
  - Focus on correctness - extensive unit test suite

- Coming soon:
  - Finite Volume and Finite difference functionality
  - Polygons and polyhedra
  - “user friendly” interface