

Using Intrelab to Quickly Prototype Interface Algorithms

Paul Kuberry

Sandia National Laboratories

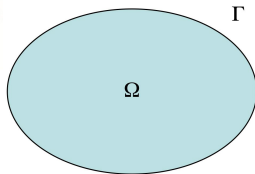
**Trilinos User Group, CSRI 90
October 29, 2014**

SAND2014-16182PE

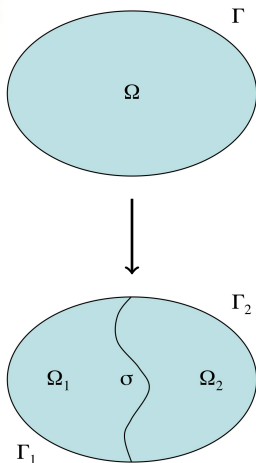
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.



Domain Layout



Domain Layout



Linear Elasticity on Ω

- Governing Equations

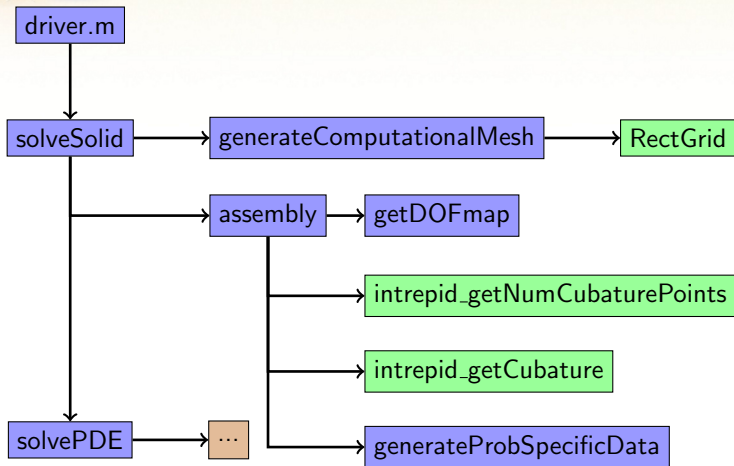
$$\rho \frac{\partial^2 \boldsymbol{\eta}}{\partial t^2} - \nabla \cdot \boldsymbol{\sigma}(\boldsymbol{\eta}) = \mathbf{f} \text{ in } \Omega$$

- We get a variational form by multiplying by a test function, integrating over Ω , and discretizing in time by the central difference scheme (explicit) and in space with \mathbb{P}^1 finite elements,

$$\int_{\Omega} \rho \frac{(\boldsymbol{\eta}_h^{n+1} - 2\boldsymbol{\eta}_h^n + \boldsymbol{\eta}_h^{n-1}))}{\Delta t^2} \cdot \boldsymbol{\xi}_h + \boldsymbol{\sigma}(\boldsymbol{\eta}_h^n) : \nabla \boldsymbol{\xi}_h \, dx = \int_{\Omega} \mathbf{f} \cdot \boldsymbol{\xi}_h \, dx \quad \forall \boldsymbol{\xi}_h \in \mathbf{V}^h \subset \mathbf{H}^1(\Omega)$$

- To create a light framework for implementing interface algorithms we at least will require
 - Basis functions (Values, Gradients, Divergence)
 - Jacobians and their determinants
 - Quadrature rules

Workflow



```
[mesh] = RectGrid(xmin, xmax, ymin, ymax, nxint, nyint, 'Triangle');  
intrepid_getNumCubaturePoints('Triangle', cubDegree);  
intrepid_getCubature(cubPoints, cubWeights, 'Triangle', cubDegree);
```

generateProbSpecificData

intrepid_setJacobian

intrepid_setJacobianInvs

intrepid_setJacobianDets

% evaluate cell Jacobians

```
cellJacobians = zeros(spaceDim, spaceDim, numCubPoints, numCells);  
intrepid_setJacobian(cellJacobians, cubPoints, ...  
    cellNodes, 'Triangle');
```

% evaluate inverses of cell Jacobians

```
cellJacobianInvs = zeros(spaceDim, spaceDim, numCubPoints, numCells);  
intrepid_setJacobianInv(cellJacobianInvs, cellJacobians);
```

% evaluate determinants of cell Jacobians

```
cellJacobianDets = zeros(numCubPoints, numCells);  
intrepid_setJacobianDet(cellJacobianDets, cellJacobians);
```

generateProbSpecificData (Reference Cell Evaluations)

```
% evaluate basis (value, gradient)
val_at_cub_points = zeros(numCubPoints, numFields);
grad_at_cub_points = zeros(spaceDim, numCubPoints, numFields);
intrepid_getBasisValues(val_at_cub_points, cubPoints, ...
    'OPERATOR_VALUE', 'Triangle', 1);
intrepid_getBasisValues(grad_at_cub_points, cubPoints, ...
    'OPERATOR_GRAD', 'Triangle', 1);

% compute cell measures
weighted_measure = zeros(numCubPoints, numCells);
intrepid_computeCellMeasure(weighted_measure, ...
    cellJacobianDets, cubWeights);

% compute cell volumes
cell_volumes = sum(weighted_measure,1);
```

generateProbSpecificData (Transform Values)

$$\int_{Phys(\mathcal{T})} \phi_j \cdot \phi_i \, dx = \int_{Ref(\mathcal{T})} \hat{\phi}_j \cdot \hat{\phi}_i \, det(J) \, d\hat{x}$$

% transform values

```
transformed_val_at_cub_points = ...  
    zeros(numCubPoints, numFields, numCells);  
intrepid_HGRADtransformVALUE(transformed_val_at_cub_points, ...  
    val_at_cub_points);
```

% combine transformed values with measures

```
weighted_transformed_val_at_cub_points = ...  
    zeros(numCubPoints, numFields, numCells);  
intrepid_multiplyMeasure(weighted_transformed_val_at_cub_points, ...  
    weighted_measure, transformed_val_at_cub_points);
```


generateProbSpecificData (Transform Gradient Values)

$$\int_{Phys(\mathcal{T})} \nabla \phi_j : \nabla \phi_i dx = \int_{Ref(\mathcal{T})} J^{-T} \hat{\nabla} \hat{\phi}_j : J^{-T} \hat{\nabla} \hat{\phi}_i det(J) d\hat{x}$$

% transform gradients

```
transformed_grad_at_cub_points = zeros(spaceDim, numCubPoints, ...  
    numFields, numCells);  
intrepid_HGRADtransformGRAD(transformed_grad_at_cub_points, ...  
    cellJacobianInvs, grad_at_cub_points);
```

% combine transformed gradients with measures

```
weighted_transformed_grad_at_cub_points = zeros(spaceDim, ...  
    numCubPoints, numFields, numCells);  
intrepid_multiplyMeasure(weighted_transformed_grad_at_cub_points, ...  
    weighted_measure, transformed_grad_at_cub_points);
```

generateProbSpecificData (Mass Matrix)

$$\int_{Phys(\mathcal{T})} \phi_j \cdot \phi_i \, dx = \int_{Ref(\mathcal{T})} \hat{\phi}_j \cdot \hat{\phi}_i \, det(J) d\hat{x}$$

% integrate mass matrix

```
cell_mass_matrices = zeros(numFields, numFields, numCells);  
intrepid_integrate(cell_mass_matrices, ...  
    transformed_val_at_cub_points, ...  
    weighted_transformed_val_at_cub_points, 'COMP_BLAS');
```

% build global mass matrix

```
cell_mass_matrices_reshape = ...  
    reshape(cell_mass_matrices, 1, numel(cell_mass_matrices));  
mass_mat = sparse(ildxVertices,jldxVertices,  
    repmat(cell_mass_matrices_reshape',2,1));
```

generateProbSpecificData (Stiffness Matrix)

$$\int_{Phys(\mathcal{T})} \nabla \phi_j : \nabla \phi_i \, dx = \int_{Ref(\mathcal{T})} J^{-T} \hat{\nabla} \hat{\phi}_j : J^{-T} \hat{\nabla} \hat{\phi}_i \, det(J) d\hat{x}$$

% integrate stiffness matrix

```
cell_deformation_matrices = zeros(numFields, numFields, numCells);  
intrepid_integrate(cell_deformation_matrices, ...  
    transformed_grad_at_cub_points, ...  
    weighted_transformed_grad_at_cub_points, 'COMP_BLAS');
```

% build global stiffness matrix

```
cell_deformation_matrices_reshape = ...  
    reshape(cell_deformation_matrices, 1, ...  
        numel(cell_deformation_matrices));  
deformation_mat = sparse(ildxVertices, jldxVertices, ...  
    repmat(cell_deformation_matrices_reshape', 2, 1));
```

Rewriting the weak form of the problem

We now have our Mass Matrix (\mathbf{M}) and our Stiffness Matrix (\mathbf{K}) and we can rewrite

$$\int_{\Omega} \rho \frac{(\boldsymbol{\eta}_h^{n+1} - 2\boldsymbol{\eta}_h^n + \boldsymbol{\eta}_h^{n-1})}{\Delta t^2} \cdot \boldsymbol{\xi}_h + \sigma(\boldsymbol{\eta}_h^n) : \nabla \boldsymbol{\xi}_h \, dx = \int_{\Omega} \mathbf{f} \cdot \boldsymbol{\xi}_h \, dx \quad \forall \boldsymbol{\xi}_h \in \mathbf{V}^h \subset \mathbf{H}^1(\Omega)$$

as

$$\frac{\rho}{\Delta t^2} \mathbf{M} \vec{\boldsymbol{\eta}}^{n+1} = \frac{\rho}{\Delta t^2} \mathbf{M} (2\vec{\boldsymbol{\eta}}^n - \vec{\boldsymbol{\eta}}^{n-1}) - \mathbf{K} \vec{\boldsymbol{\eta}}^n + \mathbf{M} \vec{\mathbf{f}}^n$$

We use a diagonal mass matrix (nodal quadrature), and therefore we can take care of boundary conditions by using this relation for all non-Dirichlet boundary nodes, and use the prescribed value at Dirichlet boundary nodes.

solvePDE



solveSolid



solvePDE

- Calculate initial conditions
- exchange interface information
- Loop over time steps using relation

$$\frac{\rho}{\Delta t^2} \mathbf{M} \vec{\eta}^{n+1} = \frac{\rho}{\Delta t^2} \mathbf{M} (2\vec{\eta}^n - \vec{\eta}^{n-1}) - K \vec{\eta}^n + \mathbf{M} \vec{f}^n$$

- End with computeError if there is a known solution



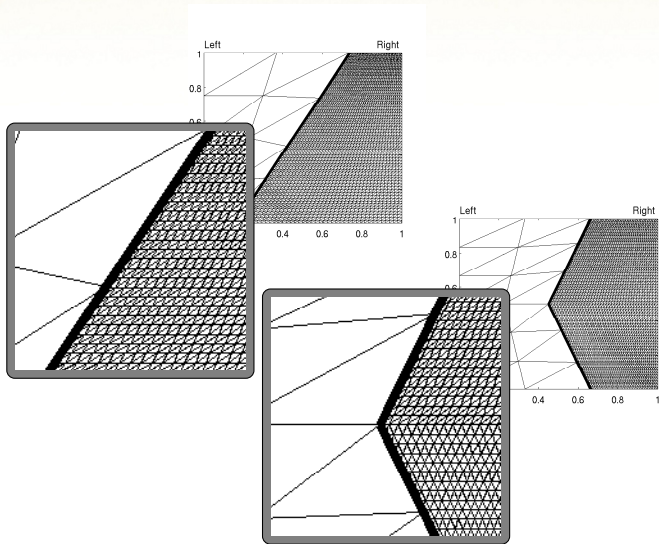
computeError

```
% set up more accurate numerical integration  
cubDegree = 6;
```

Then, get

- cubature points and weights
- Jacobians, Jacobian inverses, and determinants of Jacobians
- values and gradients on reference cells
- transformed values and gradients
- combined transformed values and gradients with measures
- ...

Computational Meshes



Convergence Rates

(Using P1 Elements)

LEFT	mesh	L2 error	L2 ratio	L2 rate	H1 error	H1 ratio	H1 rate
	0016x0010	4.45e-02	-	-	1.62e+00	-	-
	0024x0015	2.51e-02	0.56	1.42	1.10e+00	0.68	0.96
	0032x0020	1.64e-02	0.65	1.48	8.17e-01	0.74	1.04
	0048x0030	9.18e-03	0.56	1.43	5.54e-01	0.68	0.96
	0064x0040	5.56e-03	0.61	1.74	4.05e-01	0.73	1.09
	0096x0060	2.99e-03	0.54	1.53	2.70e-01	0.67	0.99
	0144x0090	1.44e-03	0.48	1.80	1.76e-01	0.65	1.06
	0208x0130	5.94e-04	0.41	2.40	1.17e-01	0.66	1.11
RIGHT	mesh	L2 error	L2 ratio	L2 rate	H1 error	H1 ratio	H1 rate
	0014x0016	8.97e-02	-	-	3.87e+00	-	-
	0021x0024	4.28e-02	0.48	1.83	2.46e+00	0.64	1.11
	0028x0032	2.68e-02	0.63	1.63	1.87e+00	0.76	0.95
	0042x0048	1.27e-02	0.47	1.85	1.23e+00	0.66	1.04
	0056x0064	7.73e-03	0.61	1.72	9.31e-01	0.76	0.96
	0084x0096	3.83e-03	0.50	1.73	6.18e-01	0.66	1.01
	0126x0144	1.71e-03	0.45	2.00	4.09e-01	0.66	1.02
	0182x0208	6.99e-04	0.41	2.42	2.79e-01	0.68	1.03
FULL	mesh	L2 error	L2 ratio	L2 rate	H1 error	H1 ratio	H1 rate
	0030x0016	7.51e-02	-	-	3.62e+00	-	-
	0045x0024	3.40e-02	0.45	1.95	2.35e+00	0.65	1.07
	0060x0032	1.93e-02	0.57	1.98	1.75e+00	0.74	1.03
	0090x0048	8.61e-03	0.45	1.99	1.16e+00	0.66	1.02
	0120x0064	4.86e-03	0.56	1.99	8.65e-01	0.75	1.01
	0180x0096	2.17e-03	0.45	1.99	5.76e-01	0.67	1.00
	0270x0144	9.72e-04	0.45	1.98	3.84e-01	0.67	1.00
	0390x0208	4.73e-04	0.49	1.96	2.66e-01	0.69	1.00

