LibMesh

Roy H. Stogner John W. Peterson

roystgnr@cfdlab.ae.utexas.edu peterson@cfdlab.ae.utexas.edu

Univ. of Texas at Austin

June 6, 2007

Outline

In this talk we will discuss:

- Goals and example applications of the libMesh library.
- Some basic steps in writing a simple libMesh application.
- Design frameworks for more complex applications.
- Adaptive Mesh Refinement.

CFDLab

Outline

In this talk we will discuss:

- Goals and example applications of the libMesh library.
- Some basic steps in writing a simple libMesh application.
- Design frameworks for more complex applications.
- Adaptive Mesh Refinement.

Outline

In this talk we will discuss:

- Goals and example applications of the libMesh library.
- Some basic steps in writing a simple libMesh application.
- Design frameworks for more complex applications.
- Adaptive Mesh Refinement.

Outline

In this talk we will discuss:

- Goals and example applications of the libMesh library.
- Some basic steps in writing a simple libMesh application.
- Design frameworks for more complex applications.
- Adaptive Mesh Refinement.

Goals

libMesh is not:

- A physics implementation.
- A stand-alone application.

libMesh is:

- A software library and toolkit.
- Objects and functions for writing parallel adaptive finite element applications.
- An interface to linear algebra, meshing, partitioning, etc. libraries.

Goals

libMesh is not:

- A physics implementation.
- A stand-alone application.

libMesh is:

- A software library and toolkit.
- Objects and functions for writing parallel adaptive finite element applications.
- An interface to linear algebra, meshing, partitioning, etc. libraries.

Goals

libMesh is not:

- A physics implementation.
- A stand-alone application.

libMesh is:

- A software library and toolkit.
- Objects and functions for writing parallel adaptive finite element applications.
- An interface to linear algebra, meshing, partitioning, etc. libraries.

CFDLab

Goals

libMesh is not:

- A physics implementation.
- A stand-alone application.

libMesh is:

- A software library and toolkit.
- Objects and functions for writing parallel adaptive finite element applications.
- An interface to linear algebra, meshing, partitioning, etc. libraries.

CFDLab

Goals

libMesh is not:

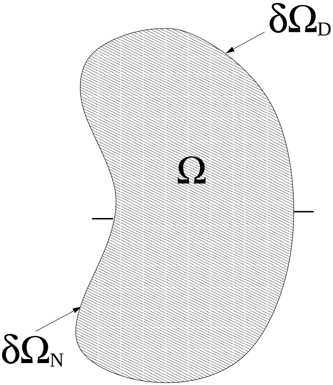
- A physics implementation.
- A stand-alone application.

libMesh is:

- A software library and toolkit.
- Objects and functions for writing parallel adaptive finite element applications.
- An interface to linear algebra, meshing, partitioning, etc. libraries.

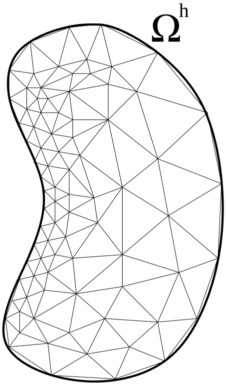
For this talk we will assume there is a mathematical model (Partial Differential Equation) to be solved in an engineering analysis:

$$
\begin{aligned}
\frac{\partial u}{\partial t} &= \mathscr{R}(u) \quad \in \Omega \\
u &= u_D \quad \in \partial\Omega_D \\
\nabla u \cdot n &= u_N \quad \in \partial\Omega_N
\end{aligned}
$$



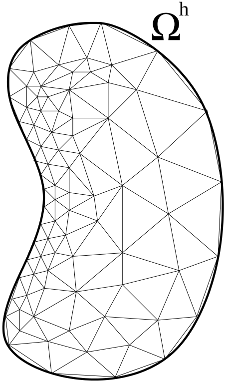
- Associated to the problem domain Ω is a libMesh data structure called a `Mesh`
- A `Mesh` is essentially a collection of finite elements

$$\Omega^h := \bigcup_e \Omega_e$$



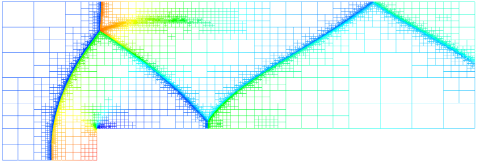
- Associated to the problem domain Ω is a libMesh data structure called a `Mesh`
- A `Mesh` is essentially a collection of finite elements

$$\Omega^h := \bigcup_e \Omega_e$$



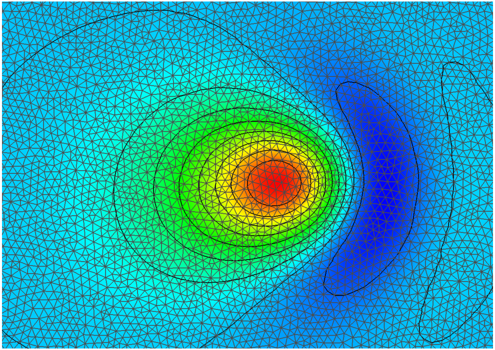
- libMesh provides some simple structured mesh generation routines as well as an interface to Triangle.

Compressible Gas Flow



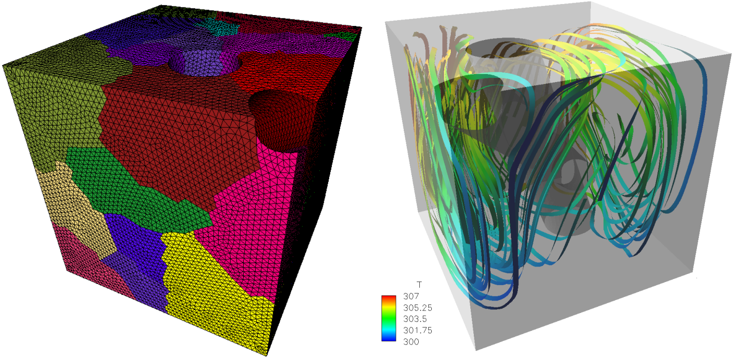
- Mach 3 flow over a forward facing step.

Shallow Water Flow



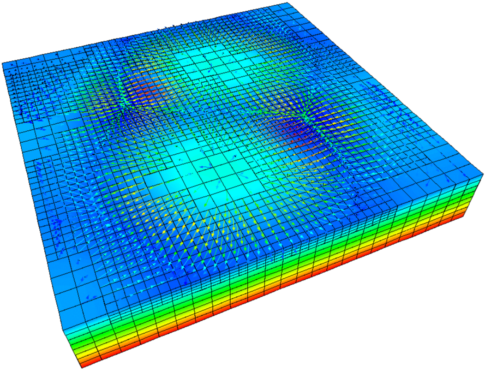
- Wave propagation from depth-averaged equations

Natural Convection



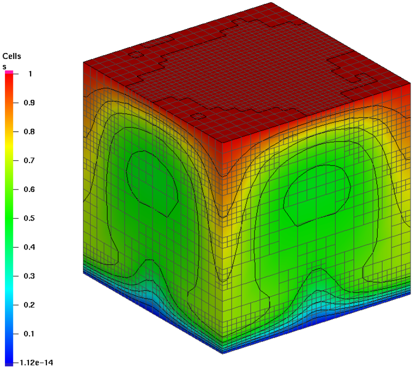
- Tetrahedral mesh of “pipe” geometry. Stream ribbons colored by temperature.

Surface-Tension-Driven Flow



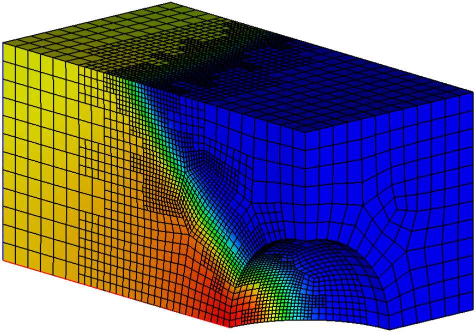
- Adaptive grid solution shown with temperature contours and velocity vectors.

Double-Diffusive Convection



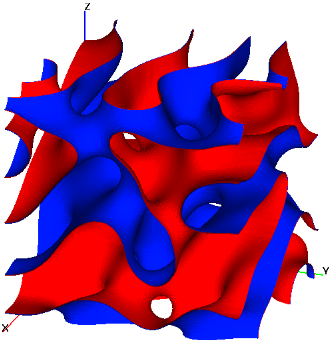
- Solute contours: a plume of warm, low-salinity fluid is convected upward through a porous medium.

Tumor Angiogenesis



- The tumor secretes a chemical which stimulates blood vessel formation.

Cahn-Hilliard Phase Decomposition



- Quenching separates fluid or alloy mixtures into multiple material phases.

- The point of departure in any FE analysis which uses LibMesh is the weighted residual statement

$$(\mathscr{R}(u), v) = 0 \qquad \forall v \in \mathcal{V}$$

- Or, more precisely, the weighted residual statement associated with the finite-dimensional space $\mathcal{V}^h \subset \mathcal{V}$

$$(\mathscr{R}(u^h), v^h) = 0 \qquad \forall v^h \in \mathcal{V}^h$$

- The point of departure in any FE analysis which uses LibMesh is the weighted residual statement

$$(\mathscr{R}(u), v) = 0 \qquad \forall v \in \mathcal{V}$$

- Or, more precisely, the weighted residual statement associated with the finite-dimensional space $\mathcal{V}^h \subset \mathcal{V}$

$$(\mathscr{R}(u^{h}), v^{h}) = 0 \qquad \forall v^{h} \in \mathcal{V}^{h}$$

Some Examples

Poisson Equation

$$-\Delta u = f \qquad \in \quad \Omega$$

Some Examples

Poisson Equation

$$-\Delta u = f \qquad \in \quad \Omega$$

Weighted Residual Statement

$$(\mathscr{R}(u), v) := \int_\Omega [\nabla u \cdot \nabla v - fv]\, dx + \int_{\partial\Omega_N} (\nabla u \cdot \boldsymbol{n})\, v\, ds$$

Some Examples

Linear Convection-Diffusion

$$-k\Delta u + \boldsymbol{b} \cdot \nabla u = f \qquad \in \quad \Omega$$

Some Examples

Linear Convection-Diffusion

$$-k\Delta u + \boldsymbol{b} \cdot \nabla u = f \qquad \in \quad \Omega$$

Weighted Residual Statement

$$(\mathscr{R}(u), v) := \int_{\Omega} [k\nabla u \cdot \nabla v + (\boldsymbol{b} \cdot \nabla u)v - fv] \, dx + \int_{\partial\Omega_N} k \left(\nabla u \cdot \boldsymbol{n}\right) v \, ds$$

Some Examples

Stokes Flow

$$\left.\begin{aligned} \nabla p - \nu \Delta \boldsymbol{u} &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= 0 \end{aligned}\right. \quad \in \quad \Omega$$

Some Examples

Stokes Flow

$$\left.\begin{aligned} \nabla p - \nu \Delta \boldsymbol{u} &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= 0 \end{aligned}\right. \quad \in \quad \Omega$$

Weighted Residual Statement

$$u := [\boldsymbol{u}, p] \quad , \quad v := [\boldsymbol{v}, q]$$

$$(\mathscr{R}(u), v) := \int_{\Omega} \left[-p \left(\nabla \cdot \boldsymbol{v} \right) + \nu \nabla \boldsymbol{u} : \nabla \boldsymbol{v} - \boldsymbol{f} \cdot \boldsymbol{v} \right.$$

$$\left. + \left(\nabla \cdot \boldsymbol{u} \right) q \right] dx + \int_{\partial \Omega_N} \left(\nu \nabla \boldsymbol{u} - p \boldsymbol{I} \right) \cdot \boldsymbol{n} \cdot \boldsymbol{v} \, ds$$

- To obtain the approximate problem, we simply replace $u \leftarrow u^h$, $v \leftarrow v^h$, and $\Omega \leftarrow \Omega^h$ in the weighted residual statement.

Poisson Equation

- For simplicity we will focus on the weighted residual statement arising from the Poisson equation, with $\partial\Omega_N = \emptyset$,

$$(\mathscr{R}(u^h), v^h) := \int_{\Omega^h} \left[\nabla u^h \cdot \nabla v^h - f v^h\right] dx = 0 \qquad \forall v^h \in \mathcal{V}^h$$

CFDLab

- The integral over Ω^h ...

$$0 = \int_{\Omega^h} \left[\nabla u^h \cdot \nabla v^h - f v^h \right] dx \quad \forall v^h \in \mathcal{V}^h$$

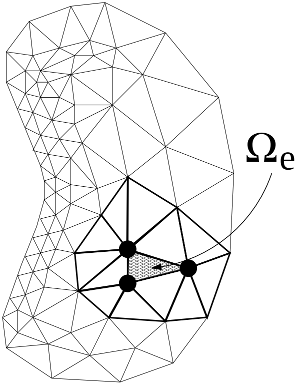
- The integral over Ω^h ... is written as a sum of integrals over the N_e finite elements:

$$
\begin{aligned}
0 &= \int_{\Omega^h} \left[\nabla u^h \cdot \nabla v^h - f v^h\right] dx \quad \forall v^h \in \mathcal{V}^h \\
&= \sum_{e=1}^{N_e} \int_{\Omega_e} \left[\nabla u^h \cdot \nabla v^h - f v^h\right] dx \quad \forall v^h \in \mathcal{V}^h
\end{aligned}
$$

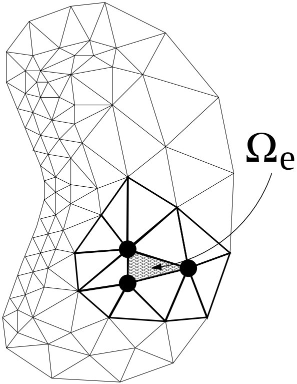
CFDLab

- An element integral will have contributions only from the global basis functions corresponding to its nodes.
- We call these local basis functions ϕ_i, $0 \le i \le N_s$.

$$v^h\big|_{\Omega_e} = \sum_{i=1}^{N_s} c_i \phi_i$$



- An element integral will have contributions only from the global basis functions corresponding to its nodes.
- We call these local basis functions ϕ_i, $0 \leq i \leq N_s$.



$$v^h\Big|_{\Omega_e} = \sum_{i=1}^{N_s} c_i \phi_i$$

$$\int_{\Omega_e} v^h \, dx = \sum_{i=1}^{N_s} c_i \int_{\Omega_e} \phi_i \, dx$$

CFDLab

- The element integrals ...

$$\int_{\Omega_e} \left[\nabla u^h \cdot \nabla v^h - f v^h \right] dx$$

- The element integrals ...

$$\int_{\Omega_e} \left[\nabla u^h \cdot \nabla v^h - f v^h\right] dx$$

- are written in terms of the local "ϕ_i" basis functions

$$\sum_{j=1}^{N_s} u_j \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \, dx - \int_{\Omega_e} f \phi_i \, dx \quad , \quad i = 1, \ldots, N_s$$

- The element integrals …

$$\int_{\Omega_e} \left[\nabla u^h \cdot \nabla v^h - f v^h \right] dx$$

- are written in terms of the local "ϕ_i" basis functions

$$\sum_{j=1}^{N_s} u_j \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \, dx - \int_{\Omega_e} f \phi_i \, dx \quad , \quad i = 1, \ldots, N_s$$

- This can be expressed naturally in matrix notation as

$$\boldsymbol{K^e U^e} - \boldsymbol{F^e}$$

- The entries of the element stiffness matrix are the integrals

$$\boldsymbol{K}^e_{ij} := \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \, dx$$

- The entries of the element stiffness matrix are the integrals

$$\boldsymbol{K}_{ij}^{e} := \int_{\Omega_e} \nabla\phi_j \cdot \nabla\phi_i \, dx$$

- While for the element right-hand side we have

$$\boldsymbol{F}_{i}^{e} := \int_{\Omega_e} f\phi_i \, dx$$

CFDLab

- The entries of the element stiffness matrix are the integrals

$$\boldsymbol{K}_{ij}^{e} := \int_{\Omega_e} \nabla\phi_j \cdot \nabla\phi_i \, dx$$

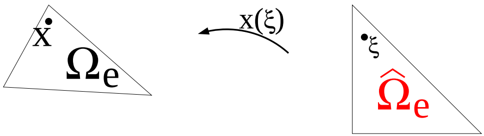
- While for the element right-hand side we have

$$\boldsymbol{F}_{i}^{e} := \int_{\Omega_e} f\phi_i \, dx$$

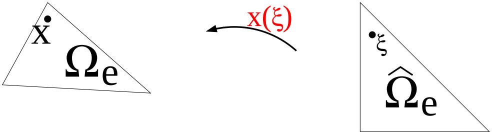
- The element stiffness matrices and right-hand sides can be “assembled” to obtain the global system of equations

$$\boldsymbol{KU} = \boldsymbol{F}$$

- The integrals are performed on a “reference” element $\hat{\Omega}_e$



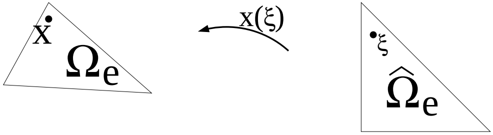
- The integrals are performed on a "reference" element $\hat{\Omega}_e$



- The Jacobian of the map $x(\xi)$ is J.

$$\boldsymbol{F}_i^e = \int_{\Omega_e} f \phi_i dx = \int_{\hat{\Omega}_e} f(x(\xi)) \phi_i |J| d\xi$$

- The integrals are performed on a “reference” element $\hat{\Omega}_e$



- Chain rule: $\nabla = J^{-1}\nabla_\xi := \hat{\nabla}_\xi$.

$$\boldsymbol{K}^e_{ij} = \int_{\Omega_e} \nabla\phi_j \cdot \nabla\phi_i \; dx = \int_{\hat{\Omega}_e} \hat{\nabla}_\xi\phi_j \cdot \hat{\nabla}_\xi\phi_i \; |J| d\xi$$

- The integrals on the “reference” element are approximated via numerical quadrature.

- The integrals on the “reference” element are approximated via numerical quadrature.
- The quadrature rule has N_q points “ξ_q” and weights “w_q”.

- The integrals on the “reference” element are approximated via numerical quadrature.
- The quadrature rule has N_q points “ξ_q” and weights “w_q”.

$$
\begin{aligned}
\boldsymbol{F}_i^e &= \int_{\hat{\Omega}_e} f \phi_i |J| d\xi \\
&\approx \sum_{q=1}^{N_q} f(x(\xi_q)) \phi_i(\xi_q) |J(\xi_q)| w_q
\end{aligned}
$$

- The integrals on the "reference" element are approximated via numerical quadrature.
- The quadrature rule has N_q points "ξ_q" and weights "w_q".

$$
\begin{aligned}
\boldsymbol{K}_{ij}^{e} &= \int_{\hat{\Omega}_e} \hat{\nabla}_\xi \phi_j \cdot \hat{\nabla}_\xi \phi_i \, |J| d\xi \\
&\approx \sum_{q=1}^{N_q} \hat{\nabla}_\xi \phi_j(\xi_q) \cdot \hat{\nabla}_\xi \phi_i(\xi_q) |J(\xi_q)| w_q
\end{aligned}
$$

- `LibMesh` provides the following variables at each quadrature point q

Code	**Math**	**Description**
`JxW[q]`	$\lvert J(\xi_q)\rvert w_q$	Jacobian times weight
`phi[i][q]`	$\phi_i(\xi_q)$	value of i^{th} shape fn.
`dphi[i][q]`	$\hat{\nabla}_\xi \phi_i(\xi_q)$	value of i^{th} shape fn. gradient
`xyz[q]`	$x(\xi_q)$	location of ξ_q in physical space

CFDLab

- The `LibMesh` representation of the matrix and rhs assembly is similar to the mathematical statements.

```
for (q=0; q<Nq; ++q)
  for (i=0; i<Ns; ++i) {
    Fe(i)    += JxW[q]*f(xyz[q])*phi[i][q];

    for (j=0; j<Ns; ++j)
      Ke(i,j) += JxW[q]*(dphi[j][q]*dphi[i][q]);
  }
```

- The `LibMesh` representation of the matrix and rhs assembly is similar to the mathematical statements.

```
for (q=0; q<Nq; ++q)
  for (i=0; i<Ns; ++i) {
    Fe(i)    += JxW[q]*f(xyz[q])*phi[i][q];

    for (j=0; j<Ns; ++j)
      Ke(i,j) += JxW[q]*(dphi[j][q]*dphi[i][q]);
  }
```

$$\boldsymbol{F}_i^e = \sum_{q=1}^{N_q} f(x(\xi_q))\phi_i(\xi_q)|J(\xi_q)|w_q$$

- The `LibMesh` representation of the matrix and rhs assembly is similar to the mathematical statements.

```
for (q=0; q<Nq; ++q)
  for (i=0; i<Ns; ++i) {
    Fe(i)    += JxW[q]*f(xyz[q])*phi[i][q];

    for (j=0; j<Ns; ++j)
      Ke(i,j) += JxW[q]*(dphi[j][q]*dphi[i][q]);
  }
```

$$\boldsymbol{F}_i^e = \sum_{q=1}^{N_q} f(x(\xi_q))\phi_i(\xi_q)|J(\xi_q)|w_q$$

- The `LibMesh` representation of the matrix and rhs assembly is similar to the mathematical statements.

```
for (q=0; q<Nq; ++q)
  for (i=0; i<Ns; ++i) {
    Fe(i)    += JxW[q]*f(xyz[q])*phi[i][q];

    for (j=0; j<Ns; ++j)
      Ke(i,j) += JxW[q]*(dphi[j][q]*dphi[i][q]);
  }
```

$$\boldsymbol{F}_i^e = \sum_{q=1}^{N_q} f(x(\xi_q))\phi_i(\xi_q)|J(\xi_q)|w_q$$

- The `LibMesh` representation of the matrix and rhs assembly is similar to the mathematical statements.

```
for (q=0; q<Nq; ++q)
  for (i=0; i<Ns; ++i) {
    Fe(i)     += JxW[q]*f(xyz[q])*phi[i][q];

    for (j=0; j<Ns; ++j)
      Ke(i,j)  += JxW[q]*(dphi[j][q]*dphi[i][q]);
  }
```

$$\boldsymbol{F}_i^e = \sum_{q=1}^{N_q} f(x(\xi_q))\phi_i(\xi_q)|J(\xi_q)|w_q$$

- The `LibMesh` representation of the matrix and rhs assembly is similar to the mathematical statements.

```
for (q=0; q<Nq; ++q)
  for (i=0; i<Ns; ++i) {
    Fe(i)    += JxW[q]*f(xyz[q])*phi[i][q];

    for (j=0; j<Ns; ++j)
      Ke(i,j) += JxW[q]*(dphi[j][q]*dphi[i][q]);
  }
```

$$\boldsymbol{K}_{ij}^{e} = \sum_{q=1}^{N_q} \hat{\nabla}_{\xi}\phi_j(\xi_q) \cdot \hat{\nabla}_{\xi}\phi_i(\xi_q) |J(\xi_q)| w_q$$

- The `LibMesh` representation of the matrix and rhs assembly is similar to the mathematical statements.

```
for (q=0; q<Nq; ++q)
  for (i=0; i<Ns; ++i) {
    Fe(i)    += JxW[q]*f(xyz[q])*phi[i][q];

    for (j=0; j<Ns; ++j)
      Ke(i,j) += JxW[q]*(dphi[j][q]*dphi[i][q]);
  }
```

$$\boldsymbol{K}_{ij}^{e} = \sum_{q=1}^{N_q} \hat{\nabla}_{\xi}\phi_j(\xi_q) \cdot \hat{\nabla}_{\xi}\phi_i(\xi_q)|J(\xi_q)|w_q$$

CFDLab

- The `LibMesh` representation of the matrix and rhs assembly is similar to the mathematical statements.

```
for (q=0; q<Nq; ++q)
  for (i=0; i<Ns; ++i) {
    Fe(i)    += JxW[q]*f(xyz[q])*phi[i][q];

    for (j=0; j<Ns; ++j)
      Ke(i,j) += JxW[q]*(dphi[j][q]*dphi[i][q]);
  }
```

$$\boldsymbol{K}_{ij}^{e} = \sum_{q=1}^{N_q} \hat{\nabla}_\xi \phi_j(\xi_q) \cdot \hat{\nabla}_\xi \phi_i(\xi_q) |J(\xi_q)| w_q$$

CFDLab

Object Oriented Programming

- Abstract Base Classes define user interfaces.
- Concrete Subclasses implement functionality.
- One physics code can work with many discretizations.

Object Oriented Programming

- Abstract Base Classes define user interfaces.
- Concrete Subclasses implement functionality.
- One physics code can work with many discretizations.

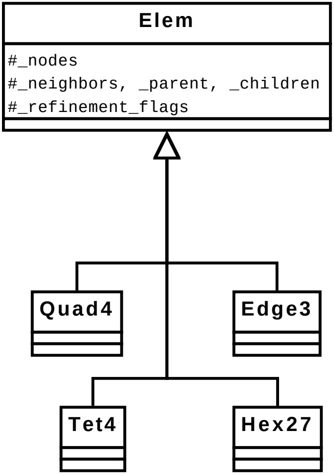
Object Oriented Programming

- Abstract Base Classes define user interfaces.
- Concrete Subclasses implement functionality.
- One physics code can work with many discretizations.

Object Oriented Programming

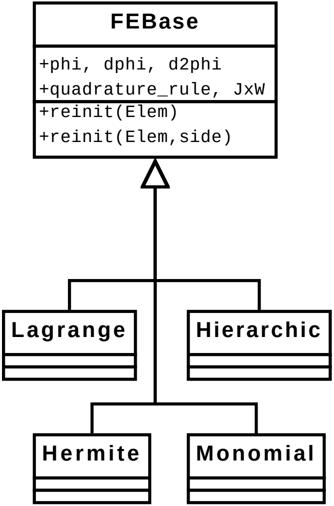
- Abstract Base Classes define user interfaces.
- Concrete Subclasses implement functionality.
- One physics code can work with many discretizations.

Geometric Element Classes



- Abstract interface gives mesh topology
- Concrete instantiations of mesh geometry
- Hides element type from most applications

Finite Element Classes



- Finite Element object builds data for each Geometric object
- User only deals with shape function, quadrature data

- For linear problems, we have already seen how the weighted residual statement leads directly to a sparse linear system of equations

$$\boldsymbol{K}\boldsymbol{U} = \boldsymbol{F}$$

- For time-dependent problems,

$$\frac{\partial u}{\partial t} = \mathscr{R}(u)$$

- we also need a way to advance the solution in time, e.g. a θ-method

$$\begin{aligned} \left(\frac{u^{n+1} - u^n}{\Delta t}, v^h \right) &= \left(\mathscr{R}(u_\theta), v^h \right) \quad \forall v^h \in \mathcal{V}^h \\ u_\theta &:= \theta u^{n+1} + (1-\theta) u^n \end{aligned}$$

- Leads to $\boldsymbol{KU} = \boldsymbol{F}$ at *each timestep*.

- For nonlinear problems, typically a sequence of linear problems must be solved, e.g. for Newton's method

$$(\mathscr{R}'(u^h)\delta u^h, v^h) = -(\mathscr{R}(u^h), v^h)$$

  where $\mathscr{R}'(u^h)$ is the linearized (Jacobian) operator associated with the PDE.
- Must solve $\boldsymbol{KU} = \boldsymbol{F}$ (Inexact Newton method) at *each iteration step*.

CFDLab

Boundary Value Problem Framework Goals

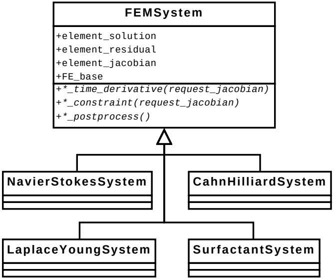
Goals:

- Improving test coverage and reliability
- Hiding of implementation details from user code
- Rapid prototyping of differential equation approximations
- Improved error estimation

Methods:

- Object-oriented System and Solver classes
- Numerical Jacobian verification

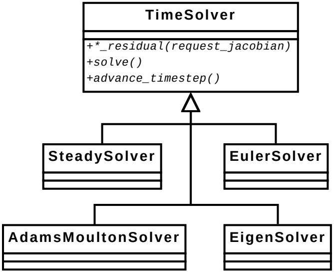
FEM System Classes



- Generalized IBVP representation
- FEMSystem does all initialization, global assembly
- User code only needs weighted time derivative and/or constraint functions

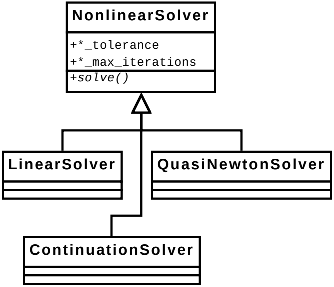
CFDLab

ODE Solver Classes



- Calls user code on each element
- Assembles element-by-element time derivatives, constraints, and weighted old solutions

Nonlinear Solver Classes



- Acquires residuals, jacobians from ODE solver
- Handles inner loops, inner solvers and tolerances, convergence tests, etc

1D refinement example

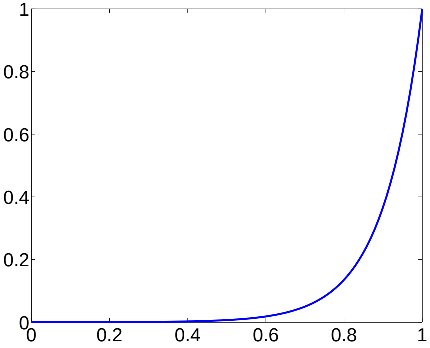
- Consider the 1D model convection-diffusion equation equation

$$\begin{cases} -u'' + bu' & = & 0 & \in & 0 \leq x \leq 1 \\ u(0) & = & 0 \\ u(1) & = & 1 \end{cases}$$

- The convection-diffusion equation can be thought of as a particularly simple form of the drift-diffusion equation.
- The exact solution is

$$u = \frac{1 - \exp(bx)}{1 - \exp(b)}$$

- For large values of b, the solution changes rapidly near $x = 1$.
- The solution for $b = 10$.

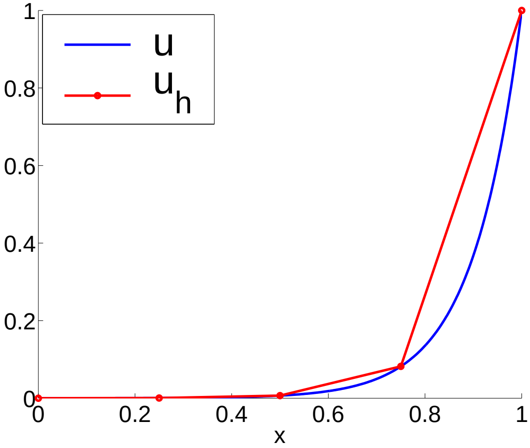


- We assume here that we have an approximate solution u_h which is the *linear interpolant* of u.
- We will measure the error between the exact solution u and the approximate solution u_h in the following (L_2) norm:

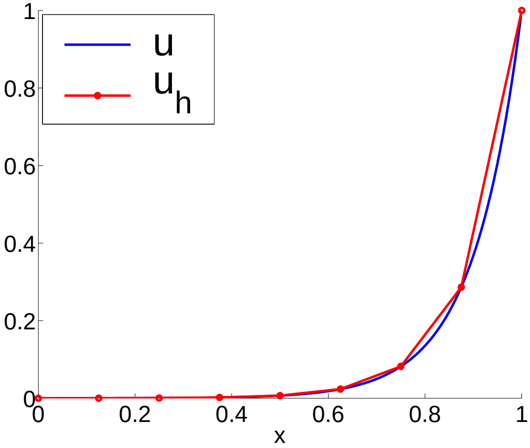
$$\begin{aligned} \|e\|_{L_2}^2 &:= \|u - u_h\|_{L_2}^2 \\ &= \int_0^1 (u - u_h)^2 \, dx \end{aligned}$$

- Consider a sequence of uniformly-refined meshes ...

CFDLab

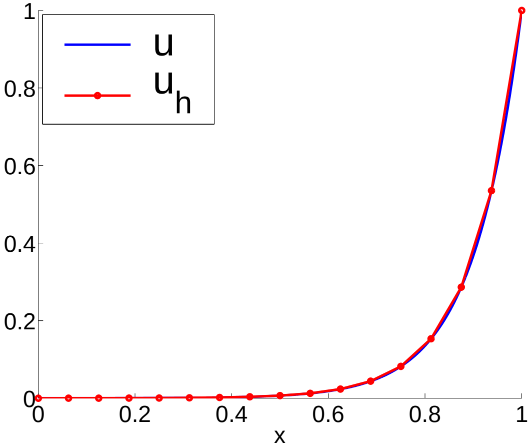


4 elements, $\|e\|_{L_2} = 0.09$



8 elements, $||e||_{L_2} = 0.027$

CFDLab



16 elements, $\|e\|_{L_2} = 0.0071$

Adaptive Refinement

- Q: Can we do "better" than uniform refinement?
- A: Yes, if we refine cells which have higher error relative to others.

A Simple Error Indicator

- In general we don't know the exact solution, and so we need a way of *estimating* the error.
- Consider the following formula for estimating the error, η, in an element defined on the interval (x_i, x_{i+1})

$$\eta^2 := \frac{h}{2} \sum_{k=i}^{i+1} [\![u'(x_k)]\!]^2$$

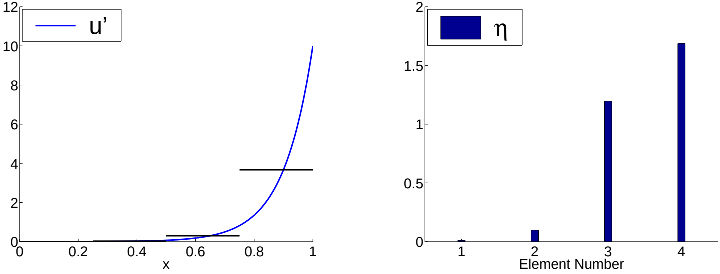
where $h := x_{i+1} - x_i$ is the element length, and

- The "jump" in u' is

$$[\![u'(x_k)]\!] := \left| u'(x_k^+) - u'(x_k^-) \right|$$

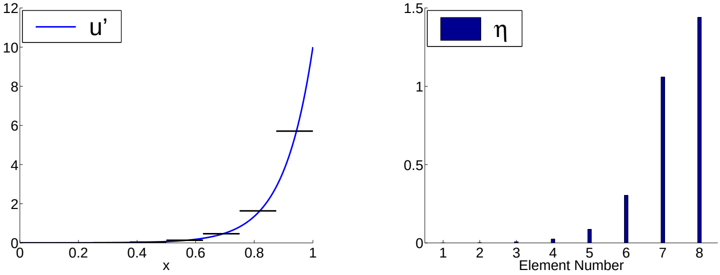
CFDLab

Error Indicator, Uniformly-Refined Grids



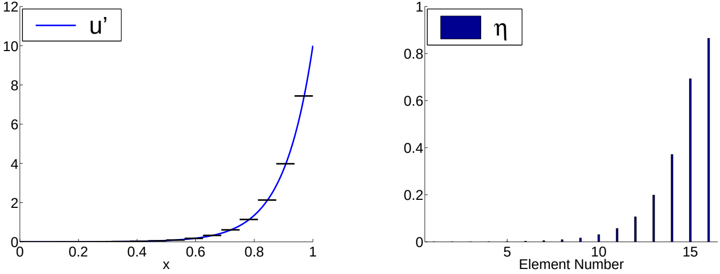
4 elements

Error Indicator, Uniformly-Refined Grids



8 elements

Error Indicator, Uniformly-Refined Grids

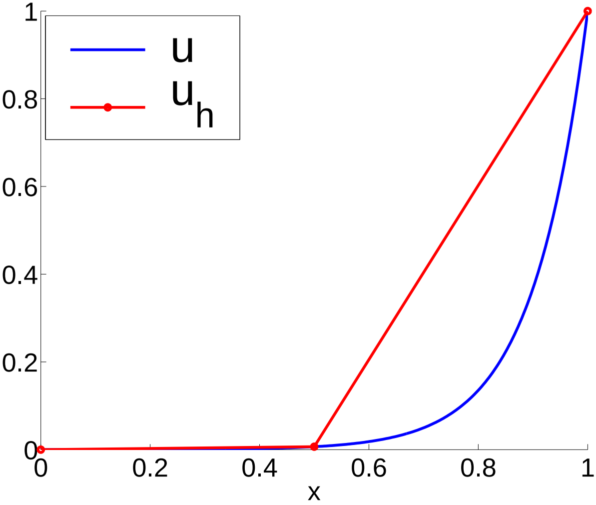


16 elements

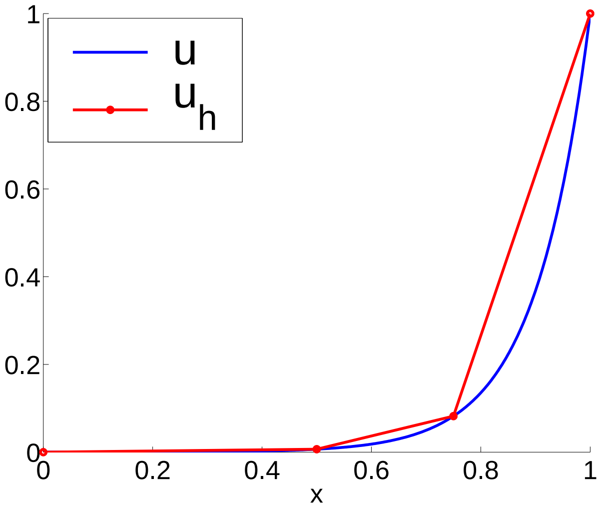
Adaptation Strategy

- A simple adaptive refinement strategy with `r_max` refinement steps for this 1D example problem is:

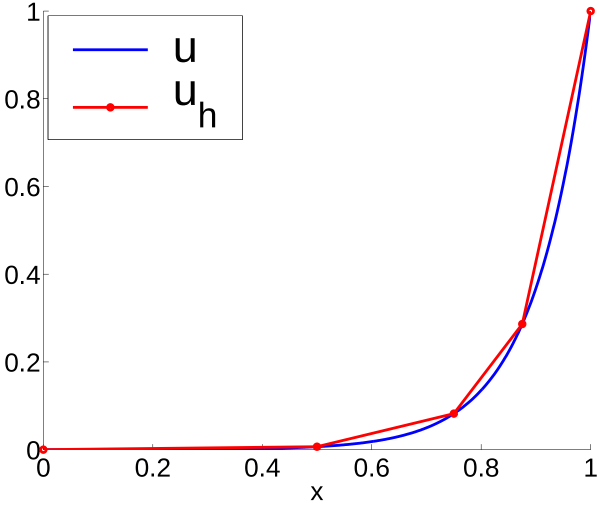
```
r=0;
while (r < r_max)
  Compute the FE solution (linear interpolant)
  Estimate the error (using flux-jump indicator)
  Refine the element with highest error
  Increment r
end
```



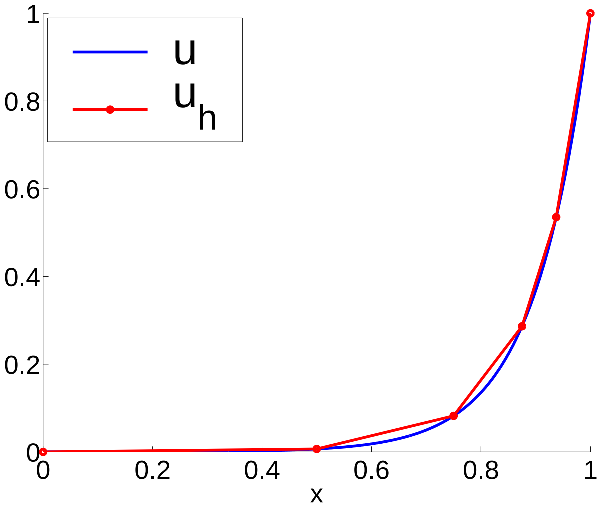
2 elements



3 elements

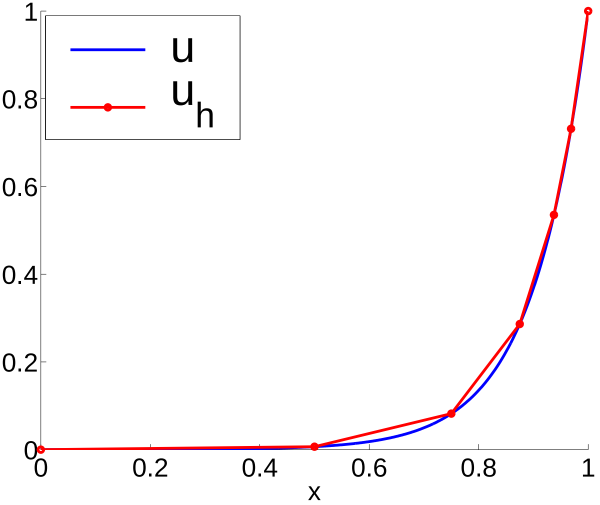


4 elements

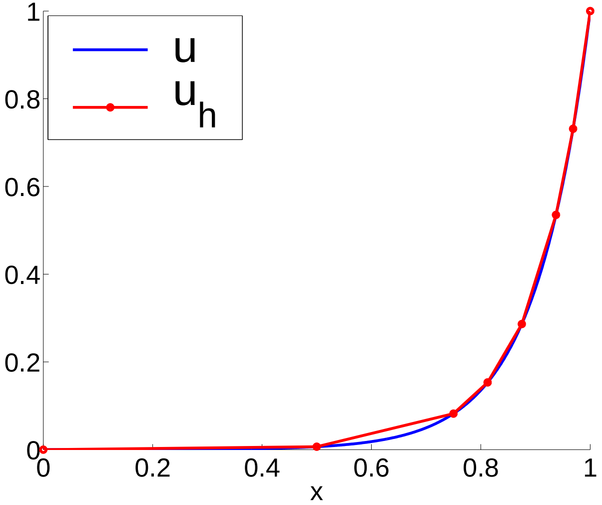


5 elements

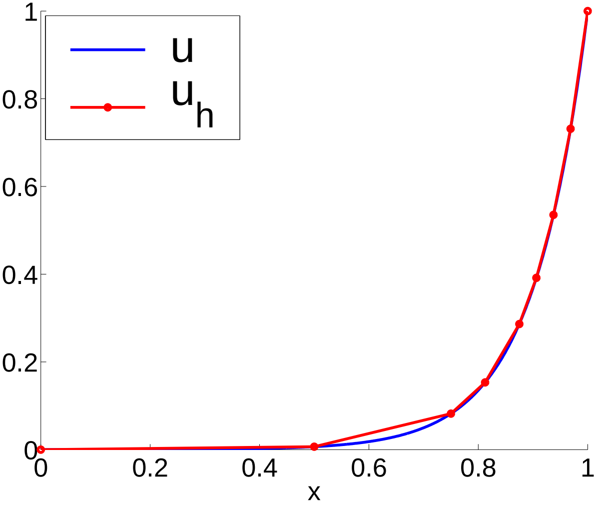
CFDLab



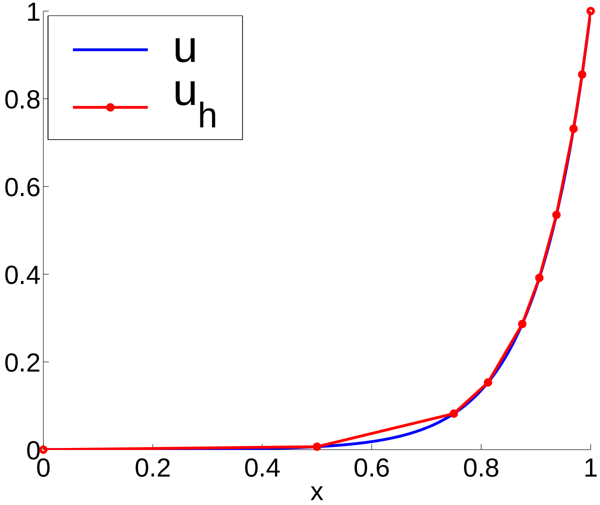
6 elements



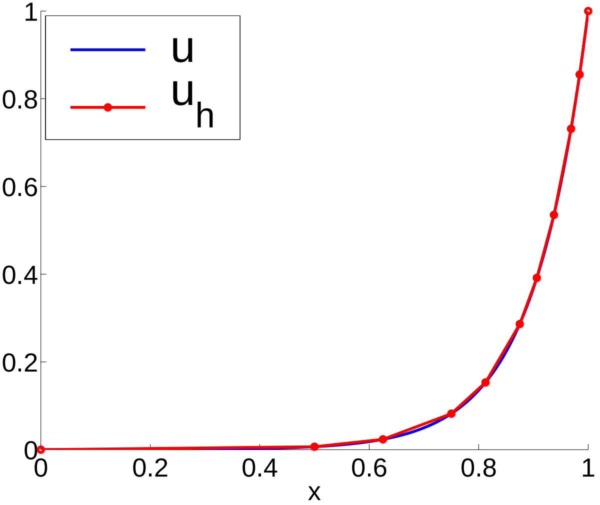
7 elements



8 elements

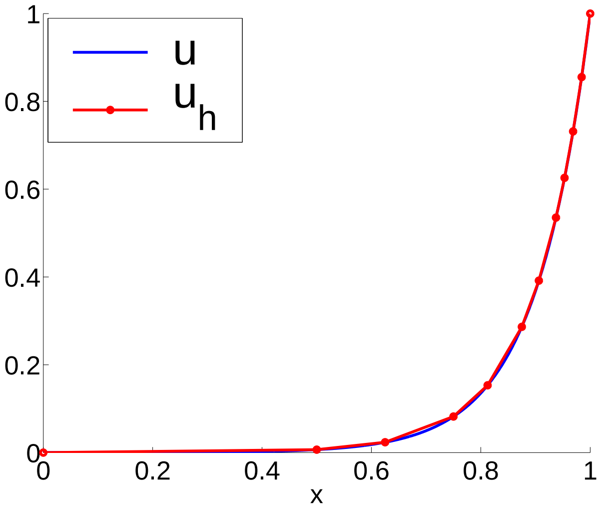


9 elements

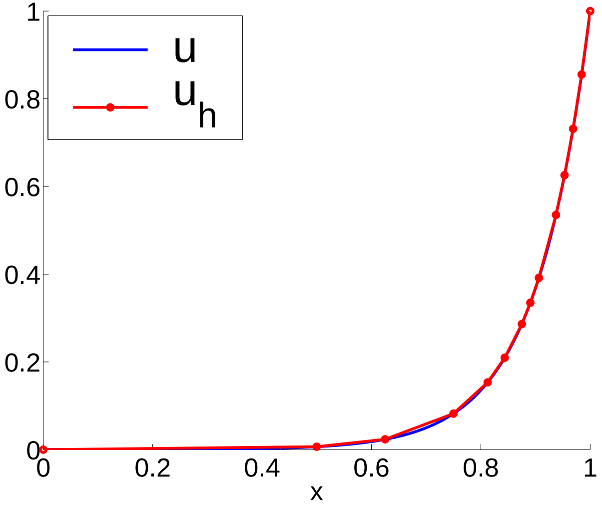


10 elements

CFDLab

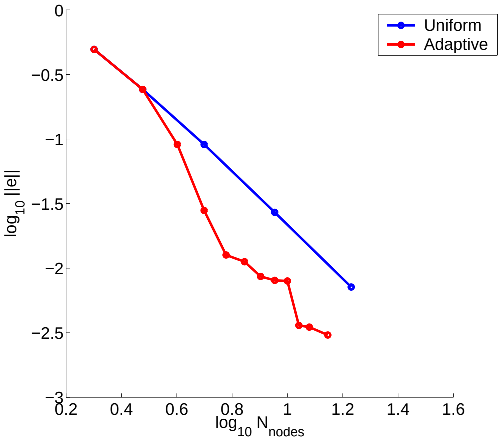


11 elements



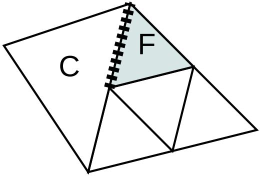
Final: 13 elements

Error Plot vs. Number of Nodes



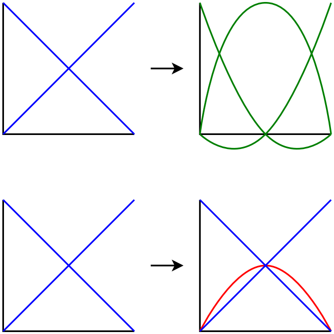
h Adaptivity Constraints

Non-conforming meshes lead to “hanging nodes”, and to provide continuous finite element function the fine element degrees of freedom must be constrained in terms of coarse element degrees of freedom.



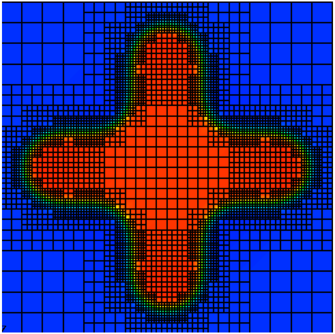
$$
\begin{aligned}
u^F &= u^C \\
\sum_i u_i^F \phi_i^F &= \sum_j u_j^C \phi_j^C \\
u_i^F &= C_{ij} u_j^C
\end{aligned}
$$

Adaptive p Constraints



- p refinement is done with hierarchic adaptivity
- Hanging degree of freedom coefficients are simply set to 0

Diffuse Interface Modeling with AMR/C



- Mesh coarsening in smooth regions is traded for mesh refinement in sharp layers
- Equivalent accuracy is achieved here with 75% fewer degrees of freedom than a uniform mesh

Installing libMesh

PETSc Download, Installation instructions, Documentation:

http://www-unix.mcs.anl.gov/petsc/petsc-2/

libMesh Downloads, Installation instructions, Documentation:

http://libmesh.sourceforge.net/

Reference

- B. Kirk, J. Peterson, R. Stogner and G. Carey, "libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations", *Engineering with Computers*, in press.

CFDLab