Parallel Adaptive PDE Simulation with `libMesh`

Roy H. Stogner

University of Texas at Austin

March 2, 2016

Outline

Outline

Background

- Modern simulation software is complex:
  - Implicit numerical methods
  - Massively parallel computers
  - Adaptive methods
  - Multiple, coupled physical processes
- There are a host of existing software libraries that excel at treating various aspects of this complexity.
- Leveraging existing software whenever possible is the most efficient way to manage this complexity.

Background

- Modern simulation software is multidisciplinary:
  - Physical Sciences
  - Engineering
  - Computer Science
  - Applied Mathematics
  - ...
- It is not reasonable to expect a single person to have all the necessary skills for developing & implementing high-performance numerical algorithms on modern computing architectures.
- Teaming is a prerequisite for success.

Background

- A large class of problems are amenable to mesh based simulation techniques.
- Consider some of the major components such a simulation:

Background

- A large class of problems are amenable to mesh based simulation techniques.
- Consider some of the major components such a simulation:
  1. Read the mesh from file
  2. Initialize data structures
  3. Construct a discrete representation of the governing equations
  4. Solve the discrete system
  5. Write out results
  6. Optionally estimate error, refine the mesh, and repeat

Background

- A large class of problems are amenable to mesh based simulation techniques.
- Consider some of the major components such a simulation:
  1. Read the mesh from file
  2. Initialize data structures
  3. Construct a discrete representation of the governing equations
  4. Solve the discrete system
  5. Write out results
  6. Optionally estimate error, refine the mesh, and repeat
- With the exception of step 3, the rest is *independent* of the class of problems being solved.

Background

- A large class of problems are amenable to mesh based simulation techniques.
- Consider some of the major components such a simulation:
  1. Read the mesh from file
  2. Initialize data structures
  3. Construct a discrete representation of the governing equations
  4. Solve the discrete system
  5. Write out results
  6. Optionally estimate error, refine the mesh, and repeat
- With the exception of step 3, the rest is *independent* of the class of problems being solved.
- This allows the major components of such a simulation to be abstracted & implemented in a reusable software library.

The libMesh Software Library

- In 2002, the libMesh library began with these ideas in mind.
- Primary goal is to provide data structures and algorithms that can be shared by disparate physical applications, that may need some combination of
  - Implicit numerical methods
  - Adaptive mesh refinement techniques
  - Parallel computing
- Unifying theme: mesh-based simulation of partial differential equations (PDEs).

The libMesh Software Library

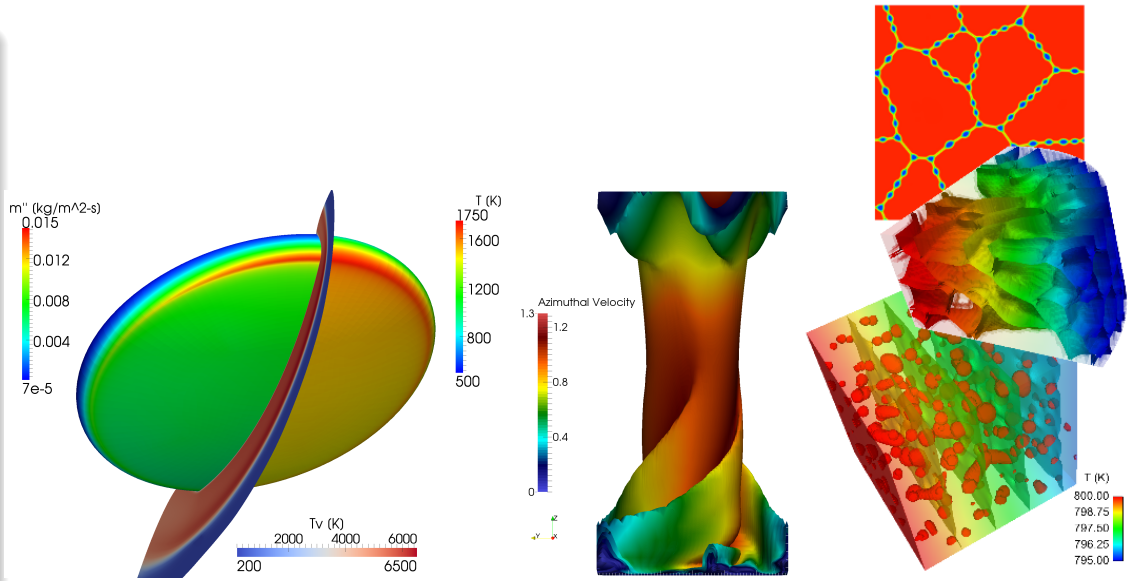
Key Point

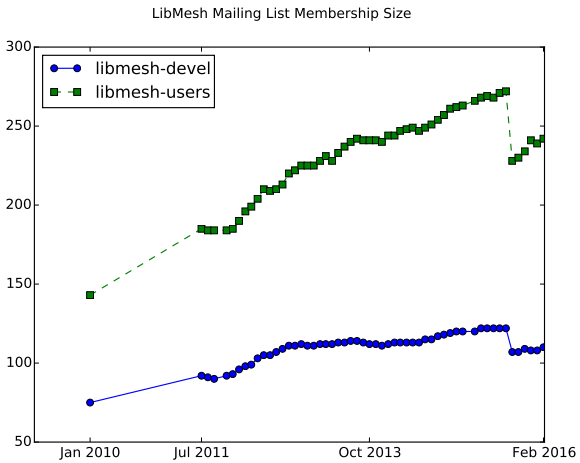
- The libMesh library is designed to be used by students, researchers, scientists, and engineers as a tool for developing simulation codes or as a tool for rapidly implementing a numerical method.
- libMesh is not an application code.
- It does not "solve problem XYZ."
  - It can be used to help you develop an application to solve problem XYZ, and to do so quickly with advanced numerical algorithms on high-performance computing platforms.

libMesh Community

Scope

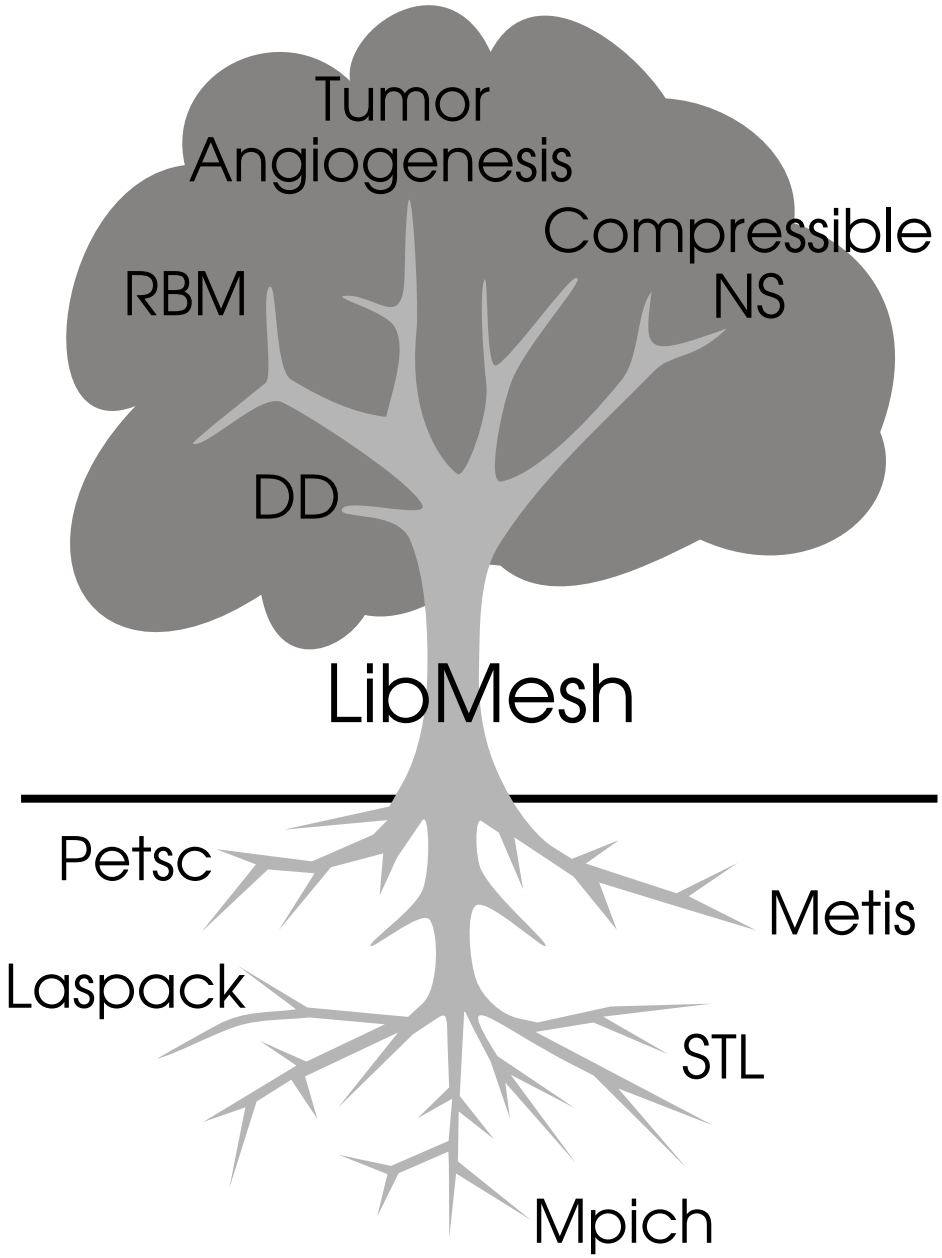
- Free, Open source
  - LGPL2 for core
- 35 Ph.D. theses, 393 papers (58 in 2015)
- ~ 10 current developers
- $160 - 240$ current users?





Challenges

- Radically different application types
- Widely dispersed core developers
  - INL, UT-Austin, U.Buffalo, JSC, MIT, Harvard, Argonne
- OSS, commercial, private applications



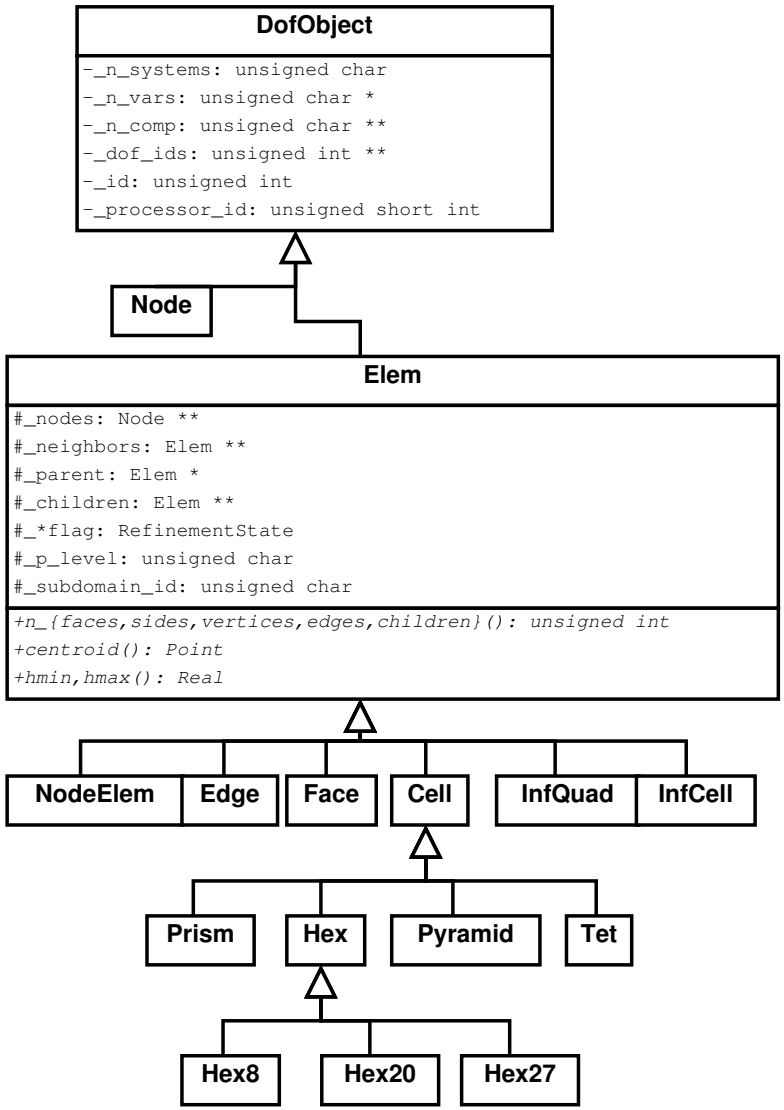
- Foundational (typically optional) library access via LibMesh's "roots".
- Application "branches" built off the library "trunk".
- Additional middleware layers (e.g. Akselos, GRINS, MOOSE) for more complex applications

Software Reusability

- At the inception of `libMesh` in 2002, there were many high-quality software libraries that implemented some aspect of the end-to-end PDE simulation process:
  - Parallel linear algebra
  - Partitioning algorithms for domain decomposition
  - Visualization formats
  - ...
- A design goal of `libMesh` has always been to provide flexible & extensible interfaces to existing software whenever possible.
- We implement the "glue" to these pieces, as well as what we viewed as the missing infrastructure:
  - Flexible data structures for the discretization of spatial domains and systems of PDEs posed on these domains.

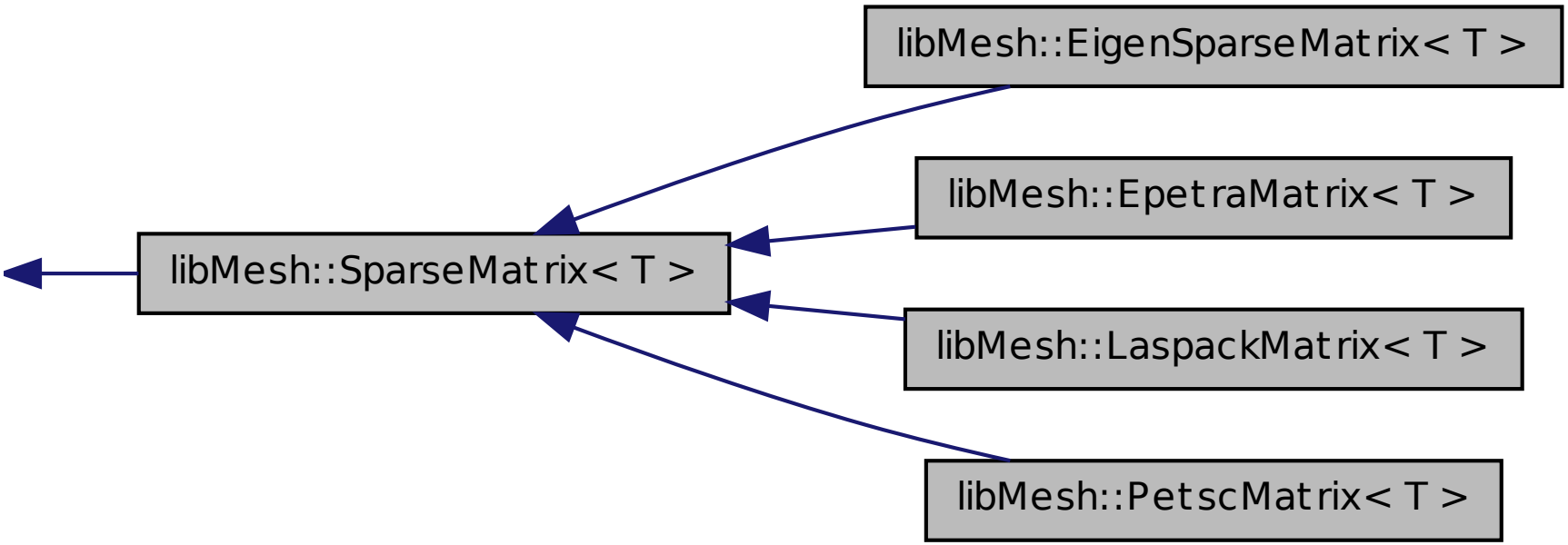
Outline

Geometric Element Classes

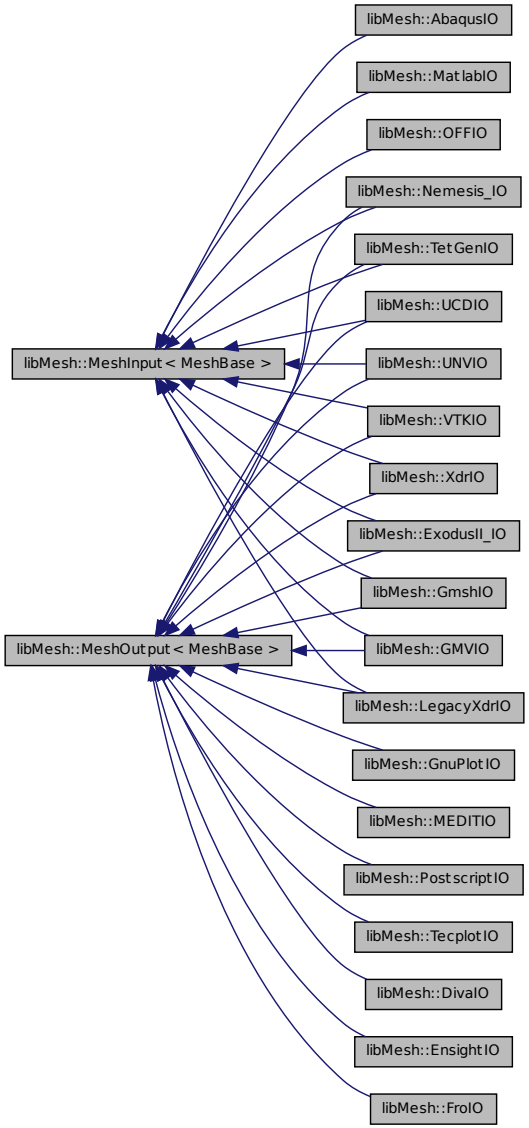


- Abstract interface gives mesh topology
- Concrete instantiations of mesh geometry
- Hides element type from most applications
- Runtime polymorphism allows mixed element types, dimensions
- Base class data arrays allow more optimization, inlining

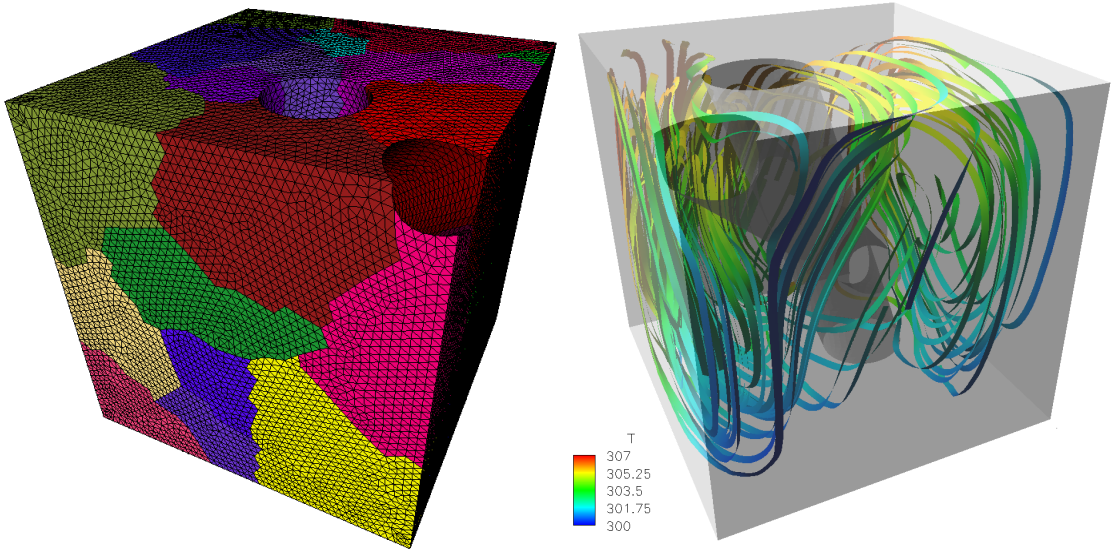
Linear Algebra

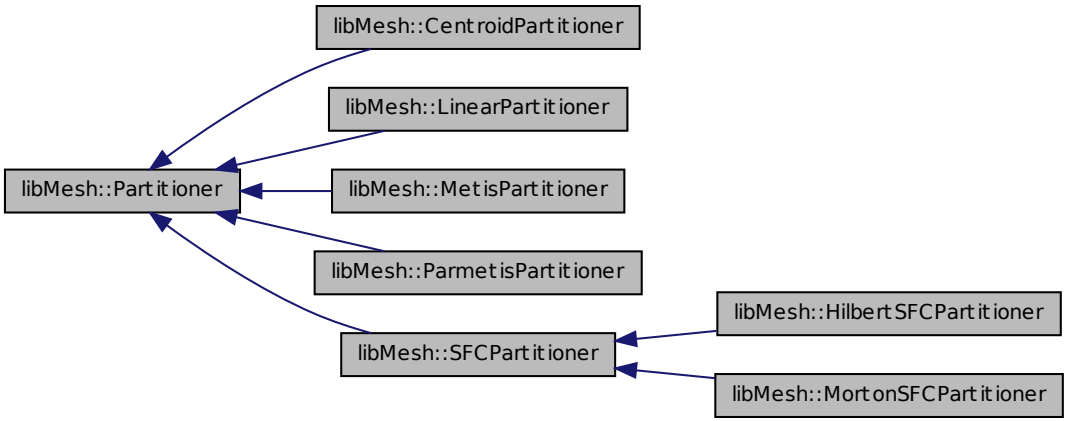


I/O formats

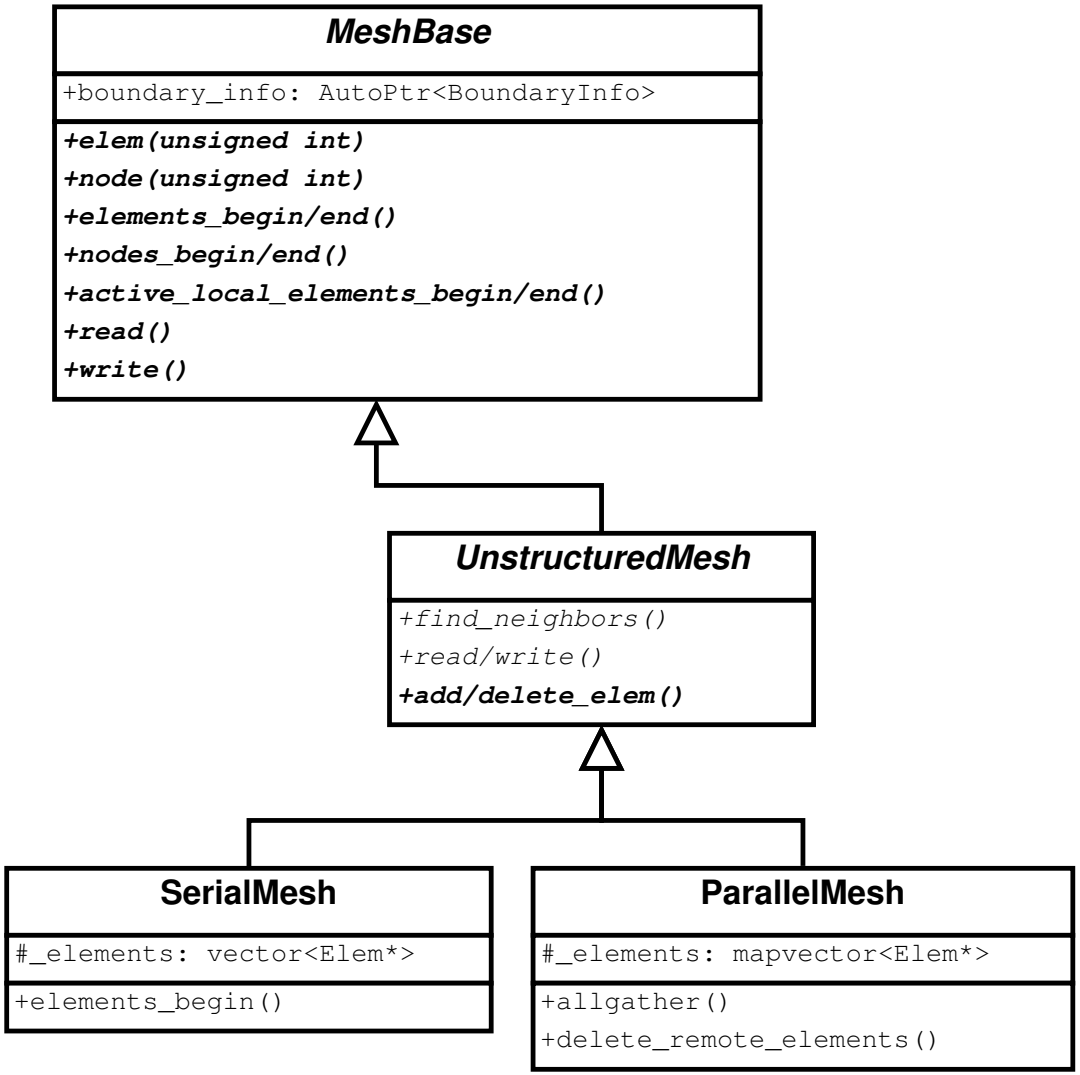

libMesh::AbaqusIO
libMesh::MatlabIO
libMesh::OFFIO
libMesh::Nemesis_IO
libMesh::TetGenIO
libMesh::UCDIO
libMesh::MeshInput< MeshBase >
libMesh::UNVIO
libMesh::VTKIO
libMesh::XdrIO
libMesh::ExodusII_IO
libMesh::GmshIO
libMesh::MeshOutput< MeshBase >
libMesh::GMVIO
libMesh::LegacyXdrIO
libMesh::GnuPlotIO
libMesh::MEDITIO
libMesh::PostscriptIO
libMesh::TecplotIO
libMesh::DivaIO
libMesh::EnsightIO
libMesh::FroIO

Domain Partitioning

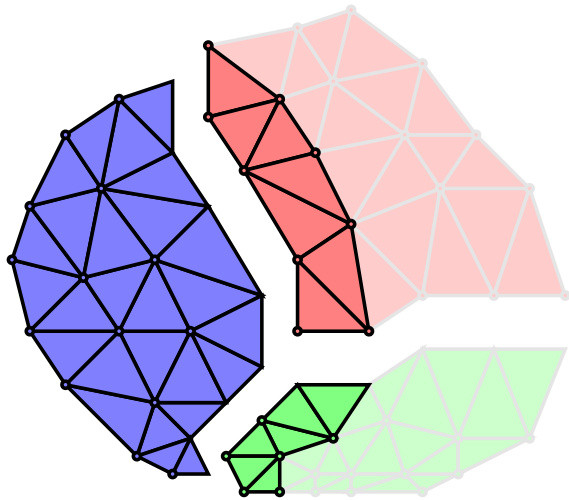




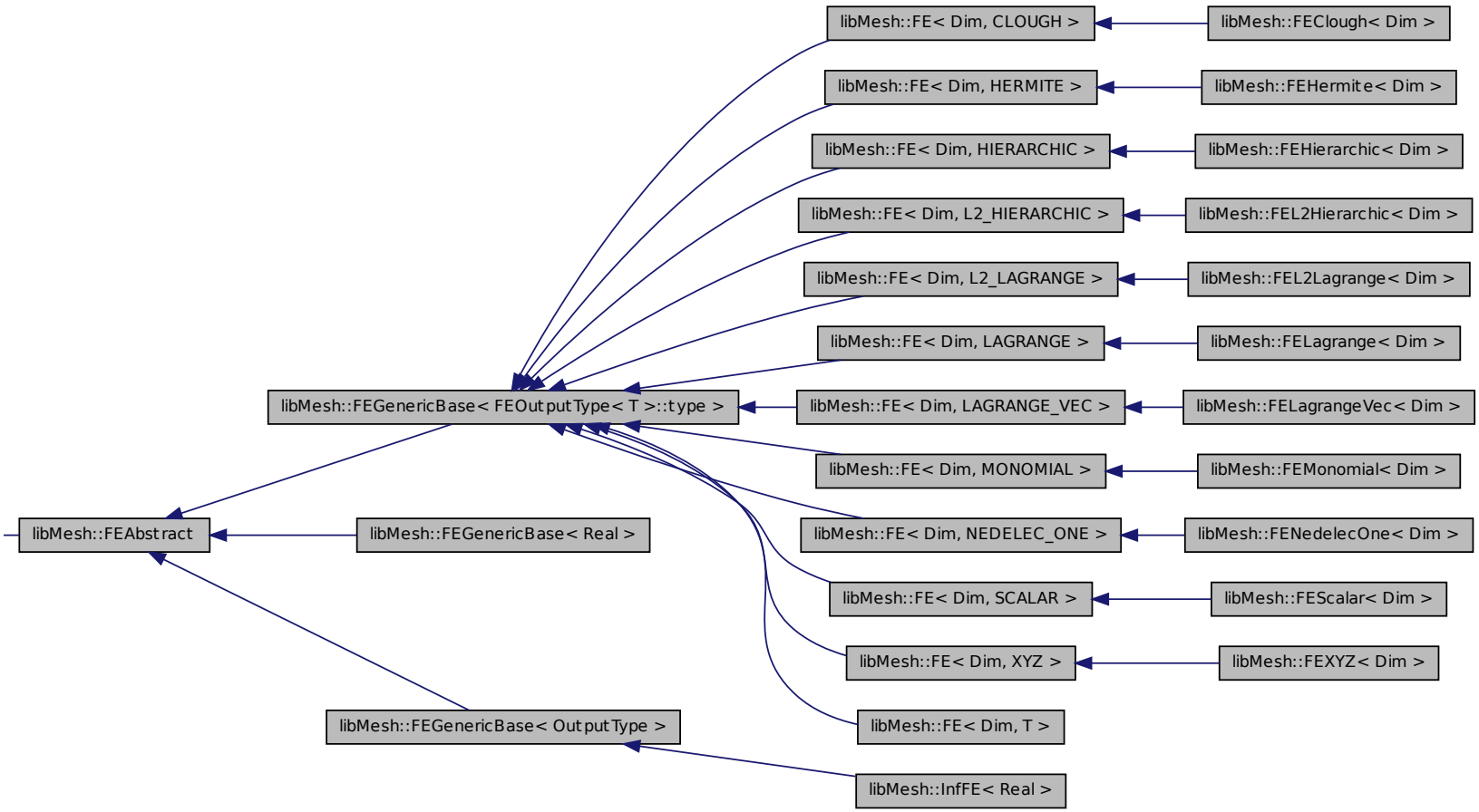
Mesh Data Structures



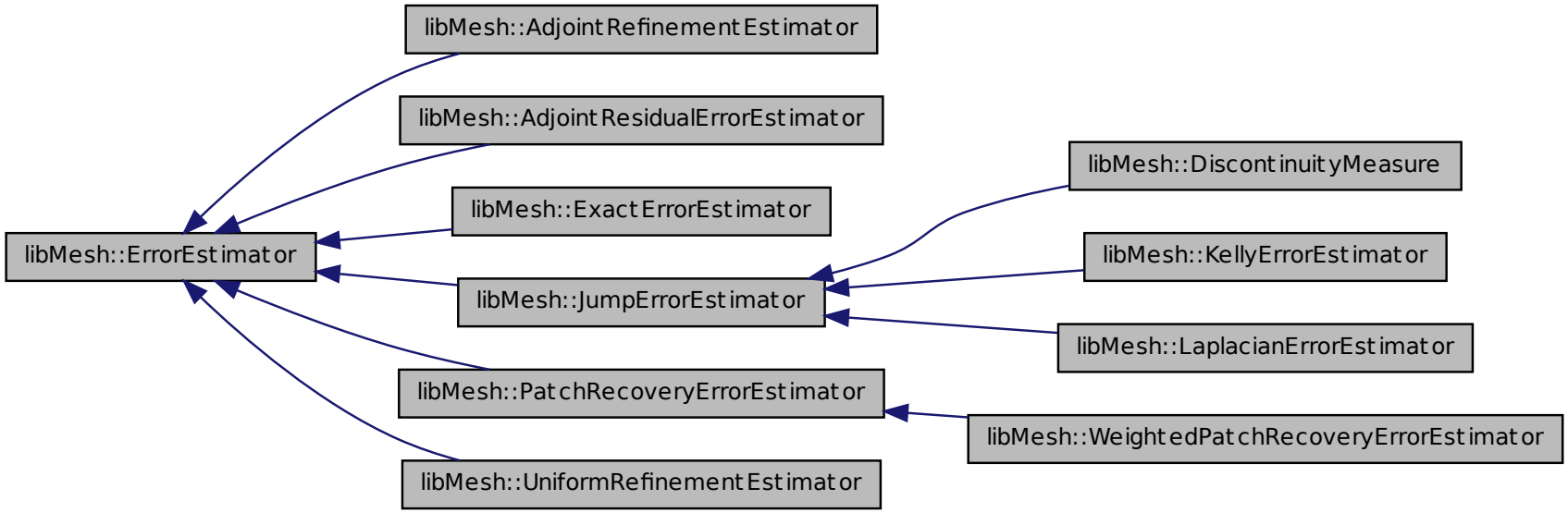
- `MeshBase` gives node or element iterators, all or active, global or local
- `SerialMesh` or `ParallelMesh` manages synchronized or distributed data



Discretization: Finite Elements



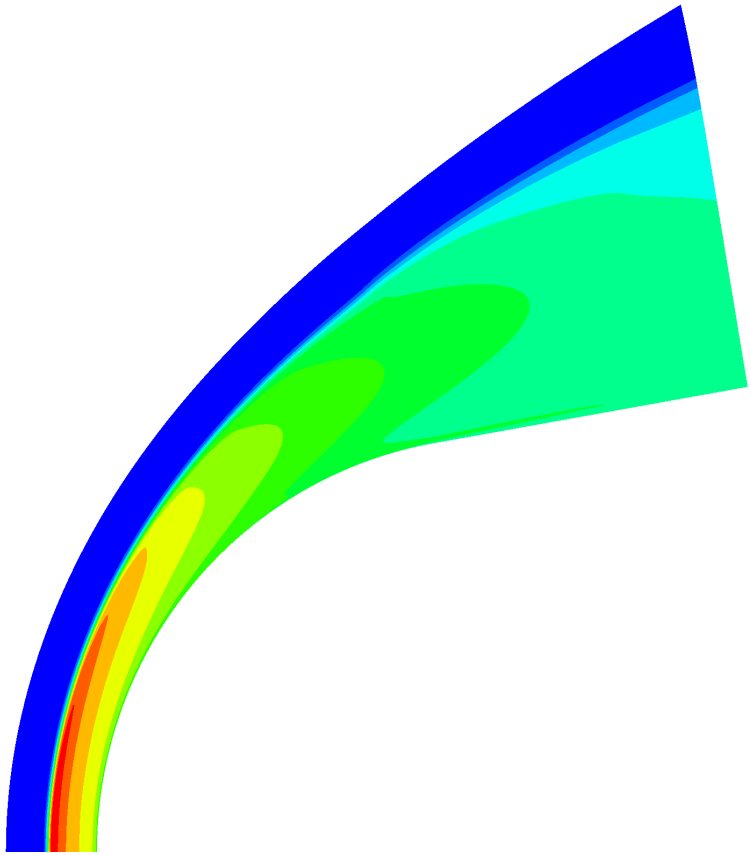
Algorithms: Error Estimation



Outline

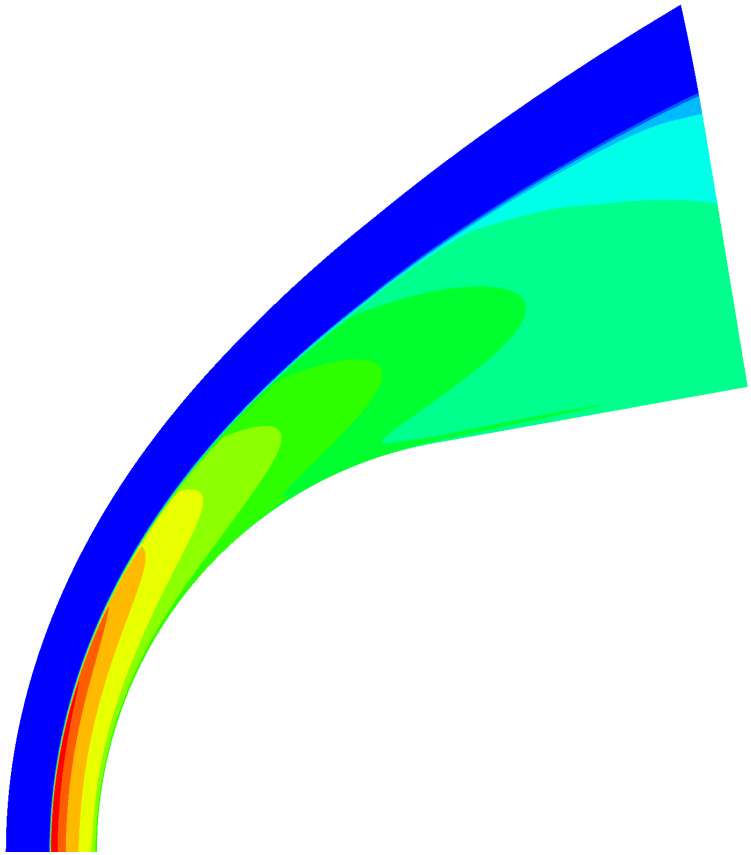
FIN-S: Fully Implicit Navier-Stokes

- Kirk, Stogner, Bauman, Oliver, Computers & Fluids, 2014
- Application for high-speed (including reentry) compressible flows in thermochemical nonequilibrium
- Including AMR capabilities
- Fully implicit coupling with surface response models



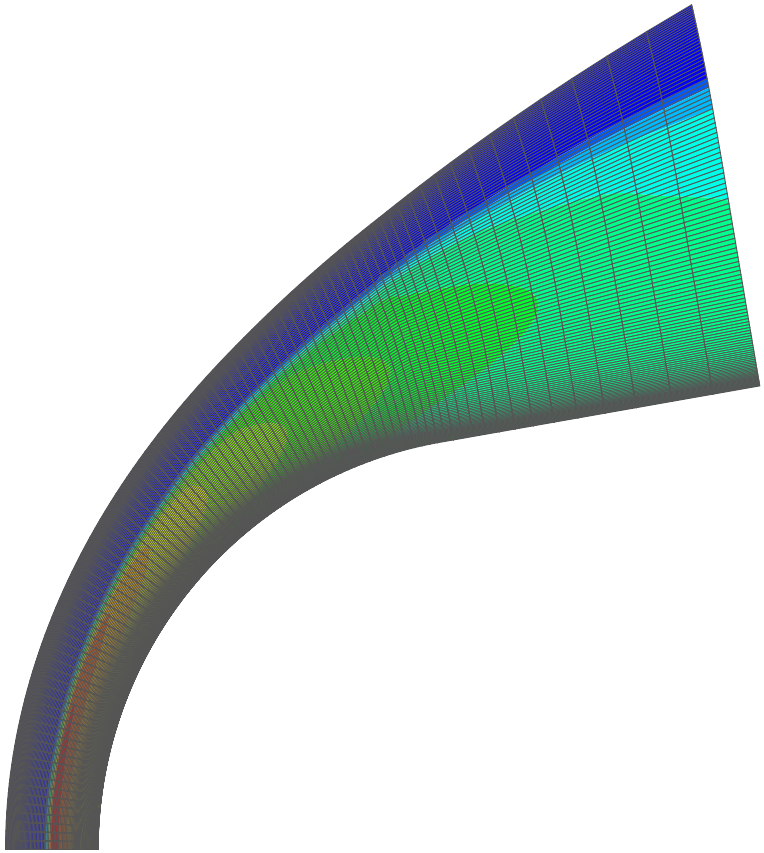
FIN-S: Fully Implicit Navier-Stokes

- Kirk, Stogner, Bauman, Oliver, Computers & Fluids, 2014
- Application for high-speed (including reentry) compressible flows in thermochemical nonequilibrium
- Including AMR capabilities
- Fully implicit coupling with surface response models



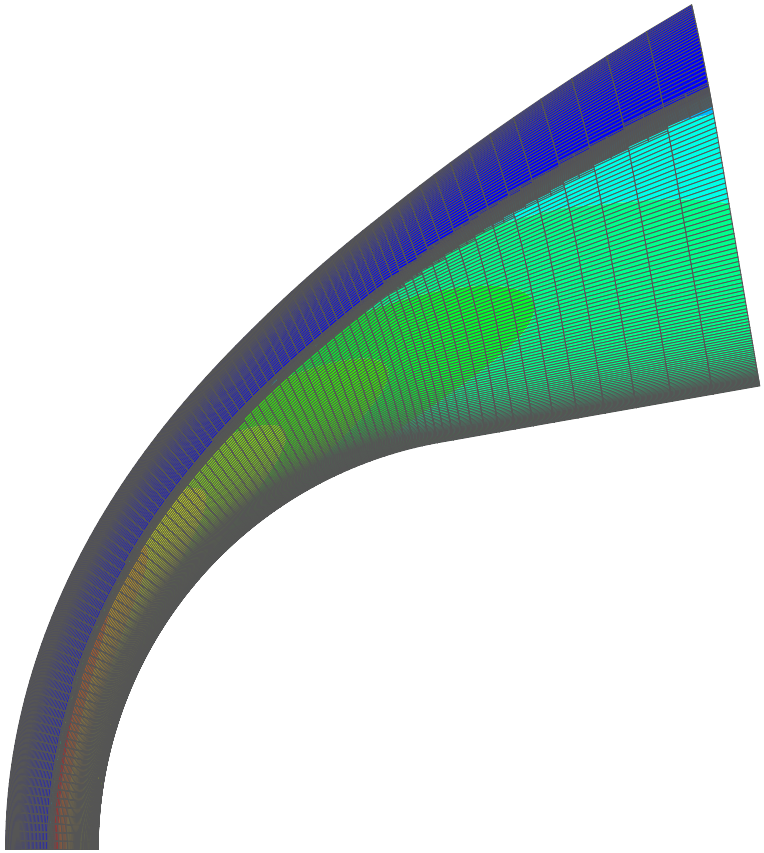
FIN-S: Fully Implicit Navier-Stokes

- Kirk, Stogner, Bauman, Oliver, Computers & Fluids, 2014
- Application for high-speed (including reentry) compressible flows in thermochemical nonequilibrium
- Including AMR capabilities
- Fully implicit coupling with surface response models



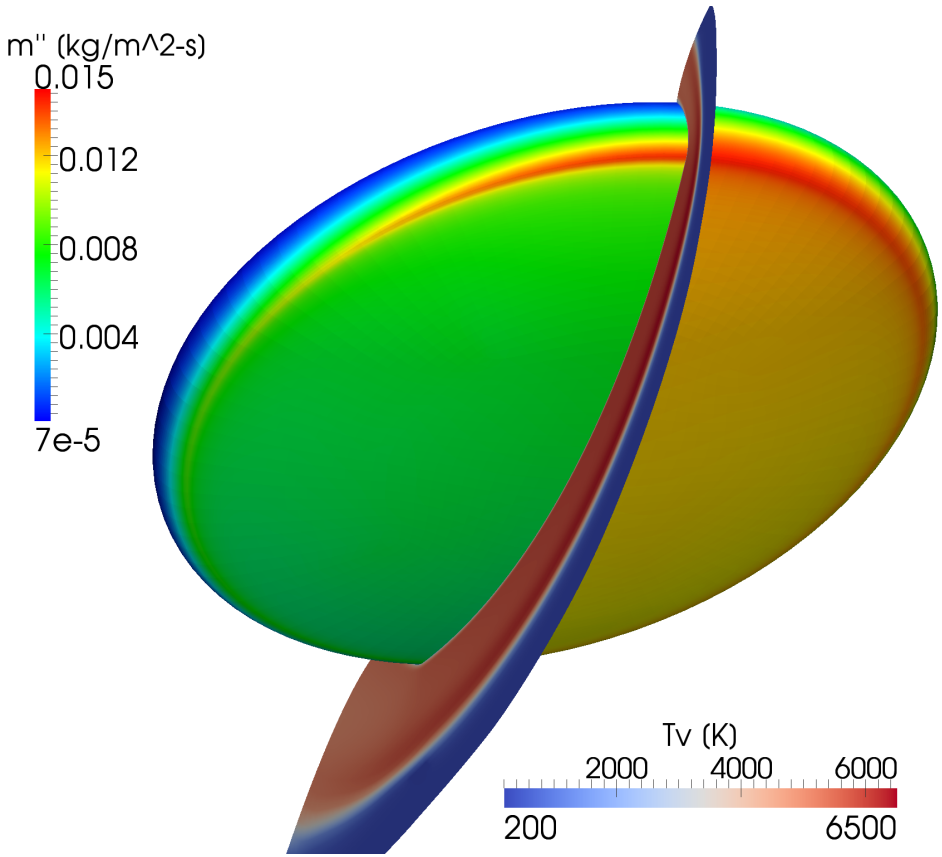
FIN-S: Fully Implicit Navier-Stokes

- Kirk, Stogner, Bauman, Oliver, Computers & Fluids, 2014
- Application for high-speed (including reentry) compressible flows in thermochemical nonequilibrium
- Including AMR capabilities
- Fully implicit coupling with surface response models



FIN-S: Fully Implicit Navier-Stokes

- Kirk, Stogner, Bauman, Oliver, Computers & Fluids, 2014
- Application for high-speed (including reentry) compressible flows in thermochemical nonequilibrium
- Including AMR capabilities
- Fully implicit coupling with surface response models



GRINS

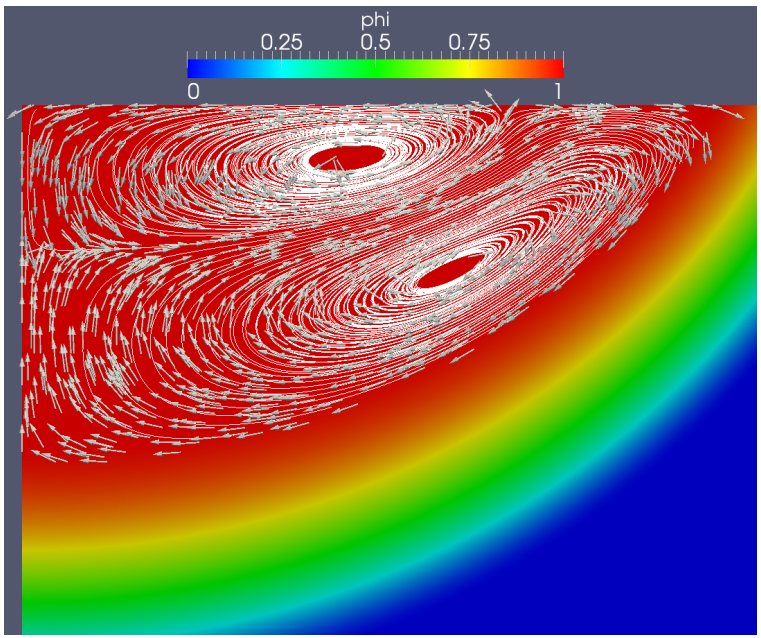
https://github.com/grinsfem/grins

- Multiphysics FEM platform built on libMesh
- Modular structure for “Physics”, solvers, QoIs, etc.
- Key feature: automatically enabled discrete adjoints (AMR, sensitivities)

GRINS

https://github.com/grinsfem/grins

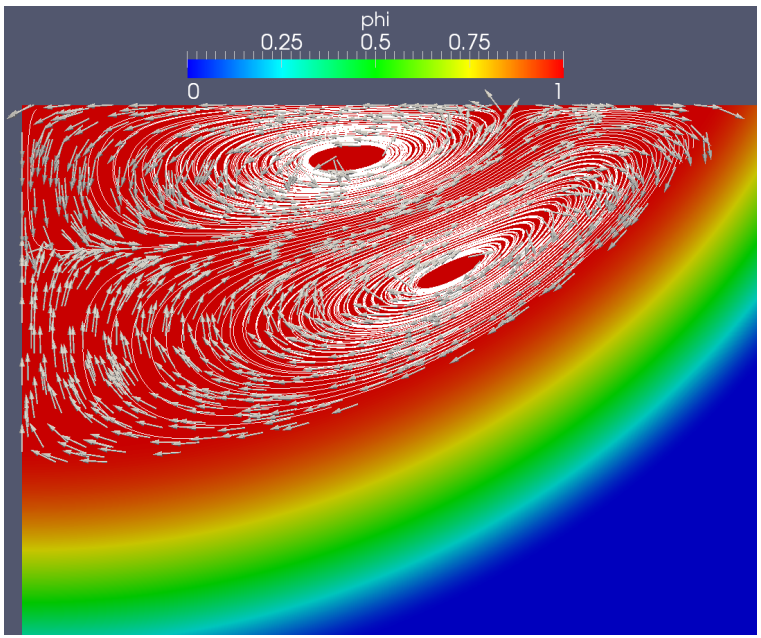
- Multiphysics FEM platform built on libMesh
- Modular structure for “Physics”, solvers, QoIs, etc.
- Key feature: automatically enabled discrete adjoints (AMR, sensitivities)

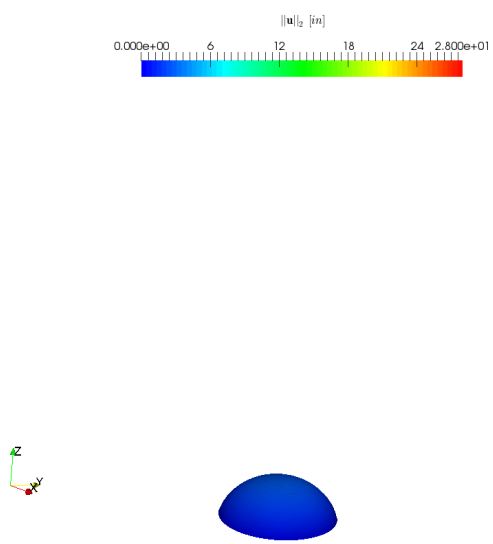


GRINS

https://github.com/grinsfem/grins

- Multiphysics FEM platform built on libMesh
- Modular structure for “Physics”, solvers, QoIs, etc.
- Key feature: automatically enabled discrete adjoints (AMR, sensitivities)

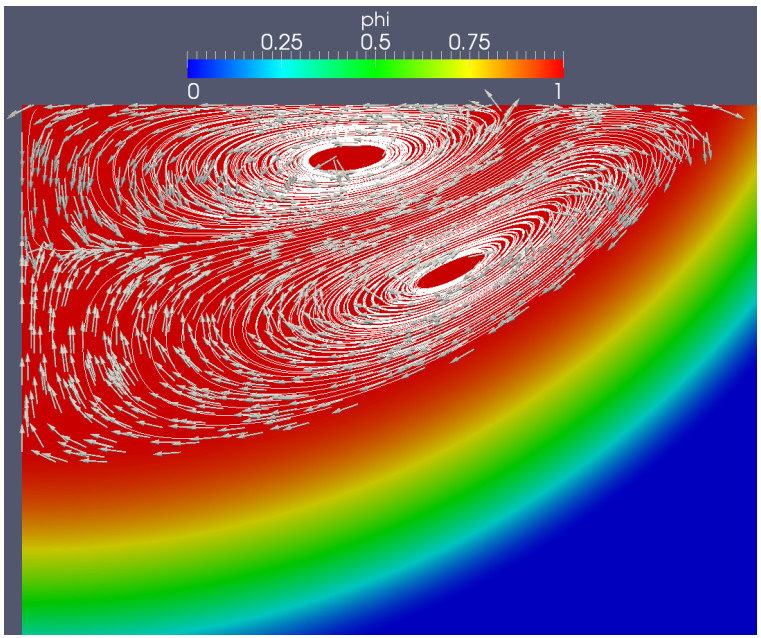


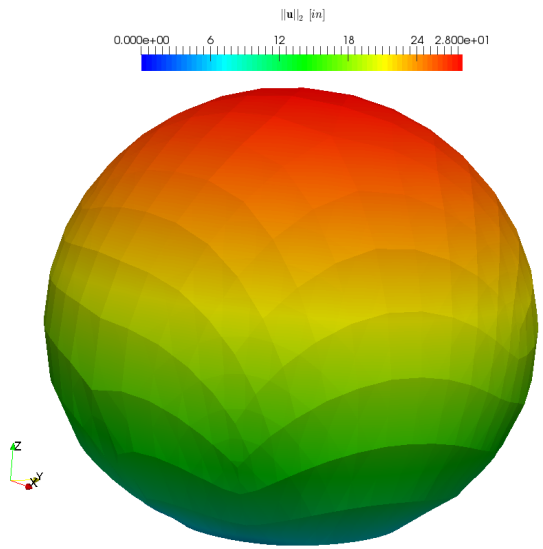


GRINS

https://github.com/grinsfem/grins

- Multiphysics FEM platform built on libMesh
- Modular structure for “Physics”, solvers, QoIs, etc.
- Key feature: automatically enabled discrete adjoints (AMR, sensitivities)

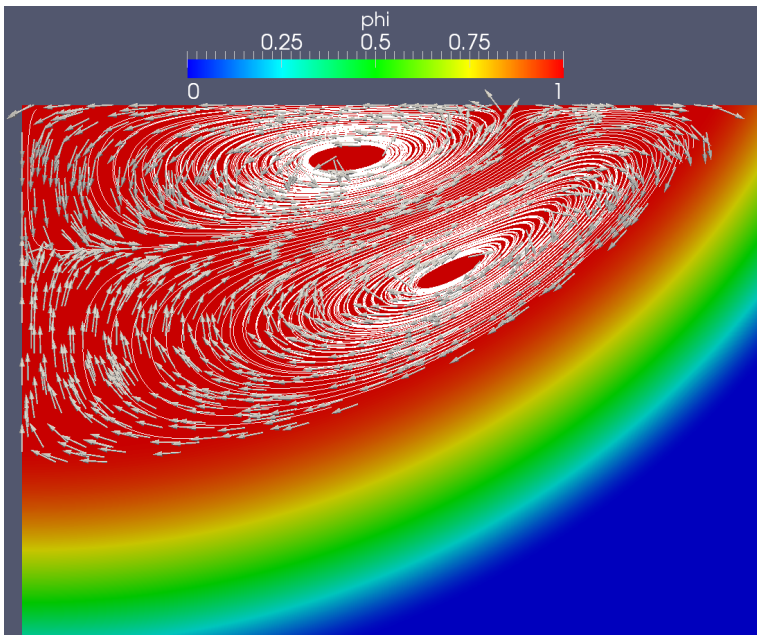


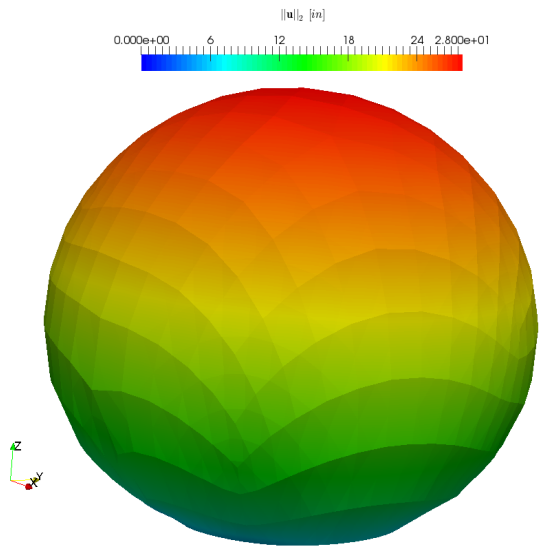


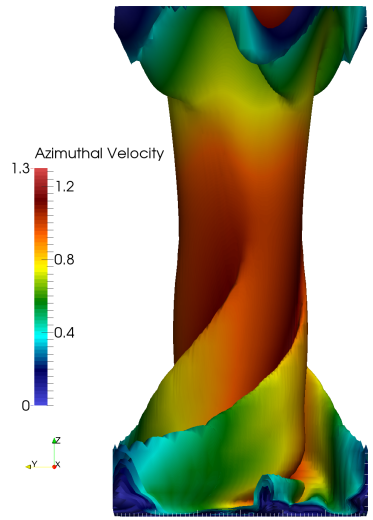
GRINS

https://github.com/grinsfem/grins

- Multiphysics FEM platform built on `libMesh`
- Modular structure for “Physics”, solvers, QoIs, etc.
- Key feature: automatically enabled discrete adjoints (AMR, sensitivities)







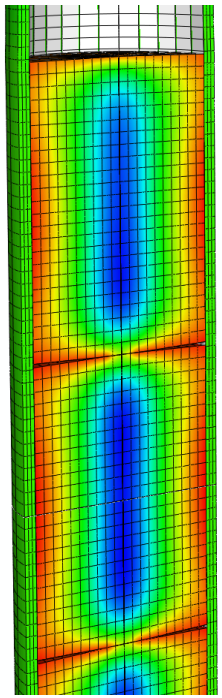
Courtesy Nick Malaya, UT Austin

The MOOSE Framework - Gaston et al., INL

Idaho National Laboratory

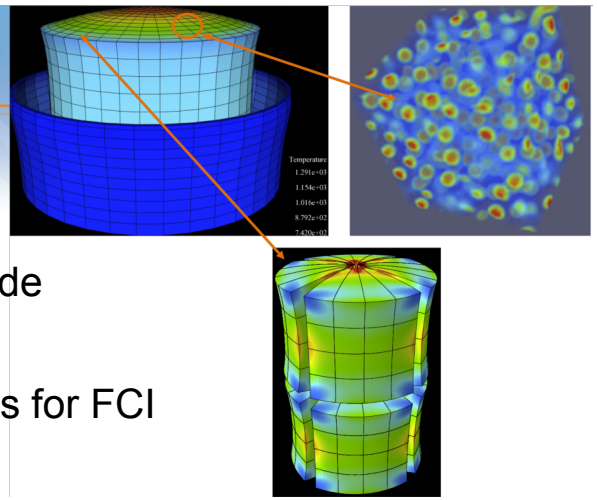
MOOSE – Multiphysics Object Oriented Simulation Environment

- A framework for solving computational nuclear engineering problems in a well planned, managed, and coordinated way
  - *Leveraged across multiple programs*
- Designed to significantly reduce the expense and time required to develop new applications
- Designed to develop analysis tools
  - *Uses very robust solution methods*
  - *Designed to be easily extended and maintained*
  - *Efficient on both a few and many processors*
- Currently supports ~7 applications which are developed and used by ~20 scientists.

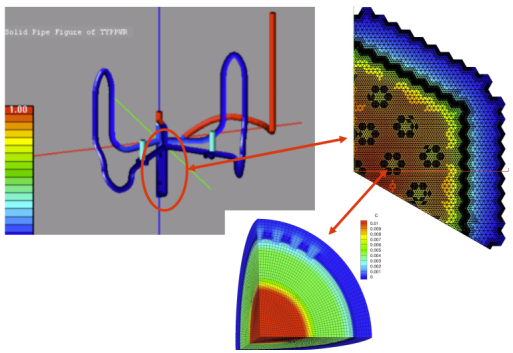


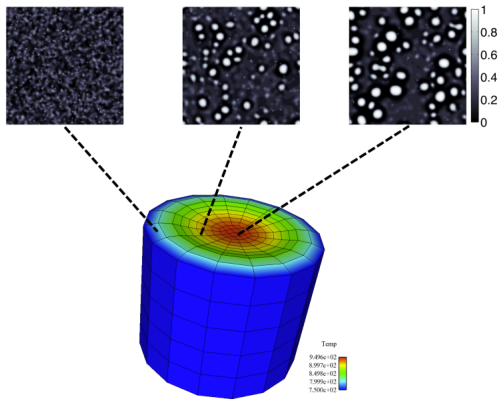
The MOOSE Framework - Gaston et al., INL

BISON fuel performance



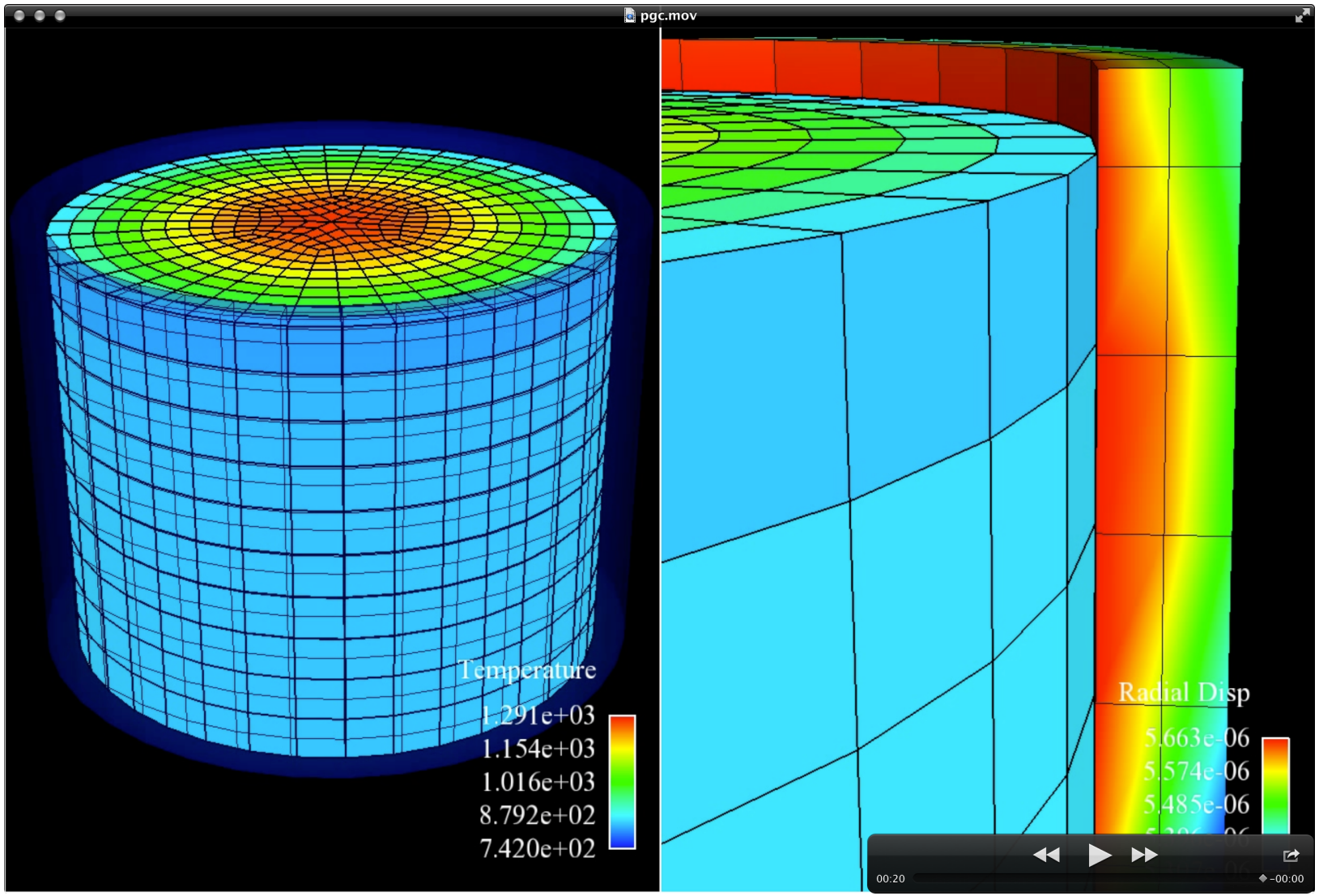
- LWR, Triso, and TRU fuel performance code
- Parallel 1D-3D thermomechanics code
- Thermal, mechanical, and chemical models for FCI
- Constituent redistribution
- Material, fission product swelling, fission gas release models
- Mesoscale-informed material models





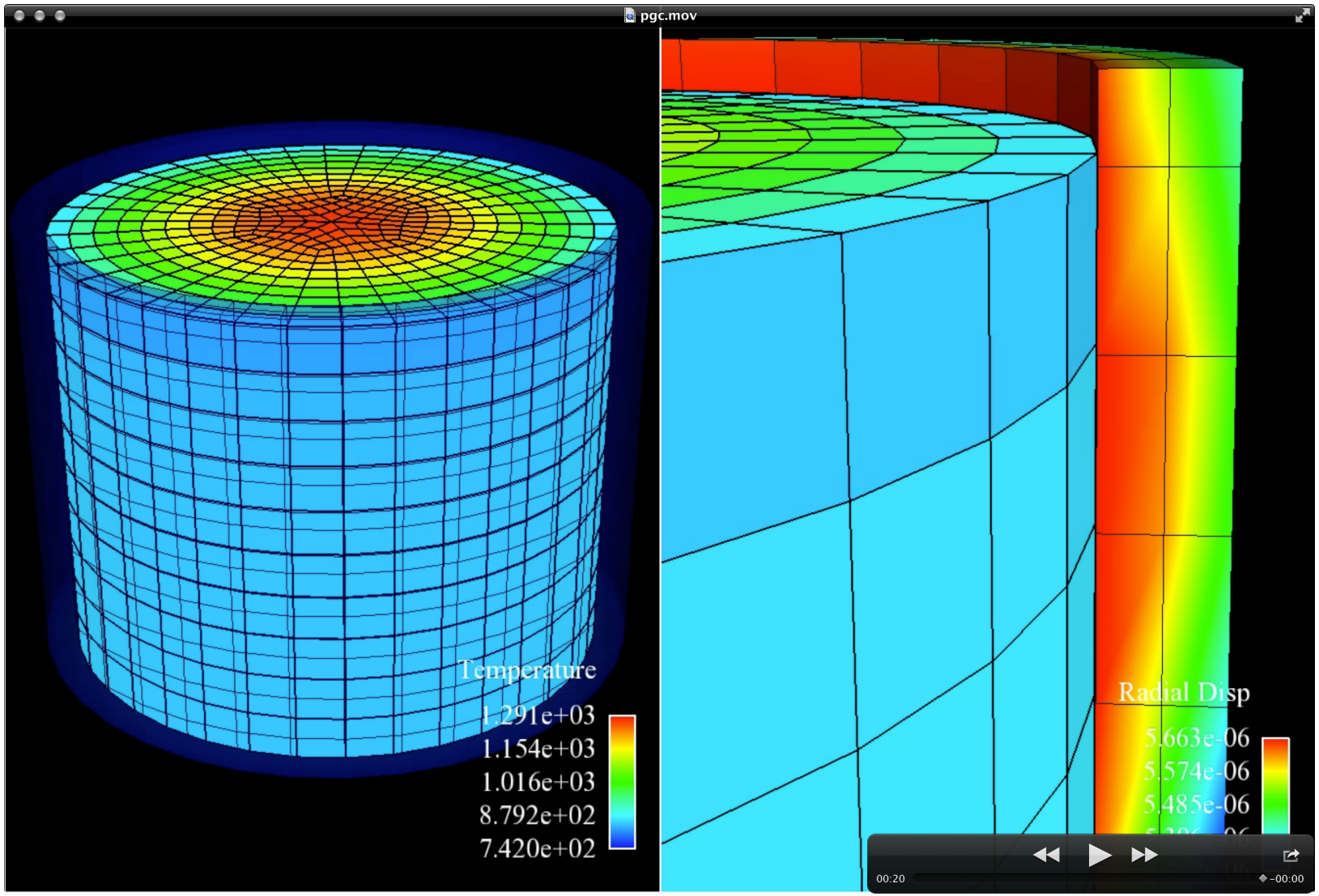
MOOSE - Coupled Thermal/Solid Mechanics

https://mooseframework.org/



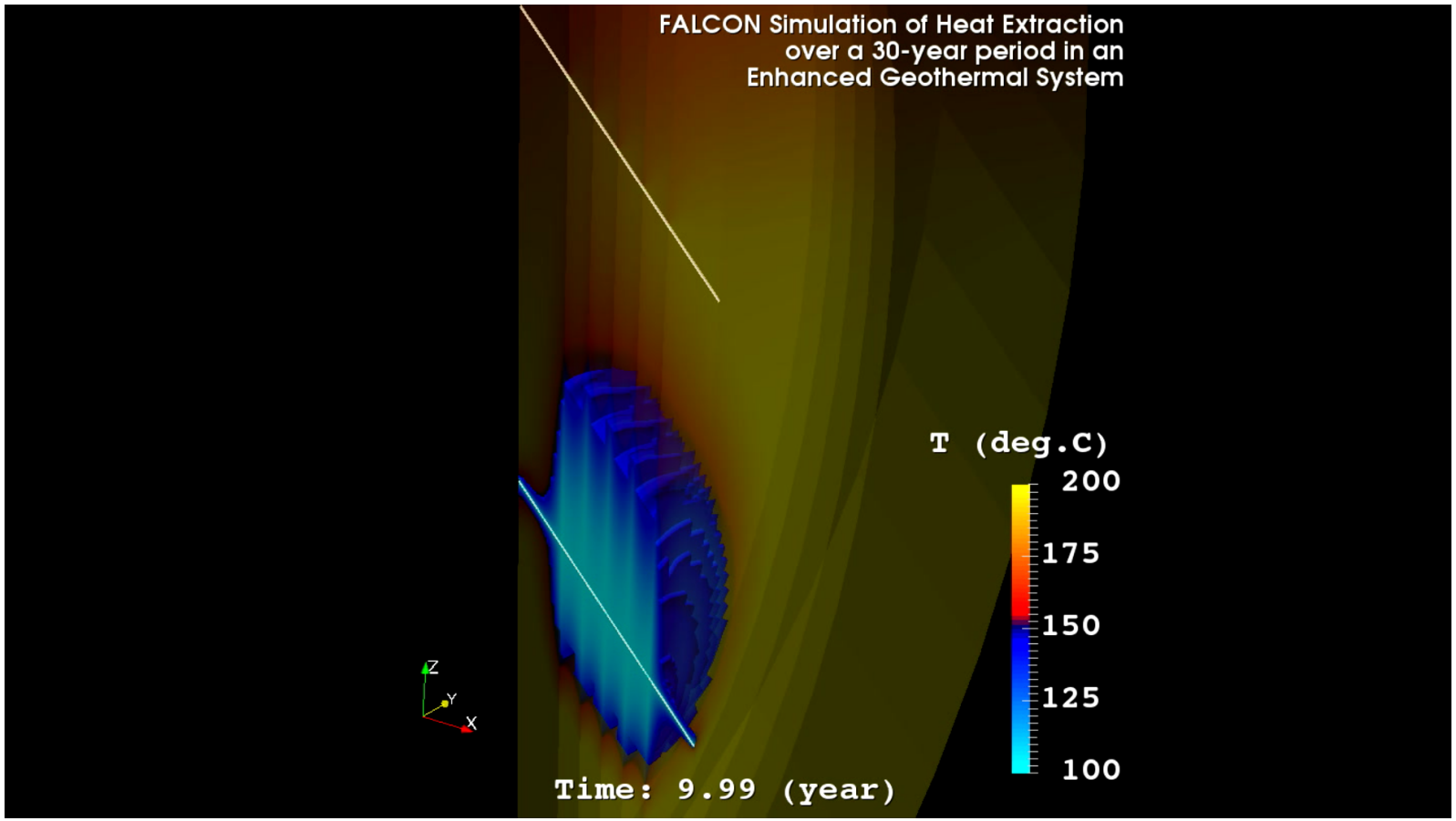
MOOSE - Coupled Thermal/Solid Mechanics

https://mooseframework.org/



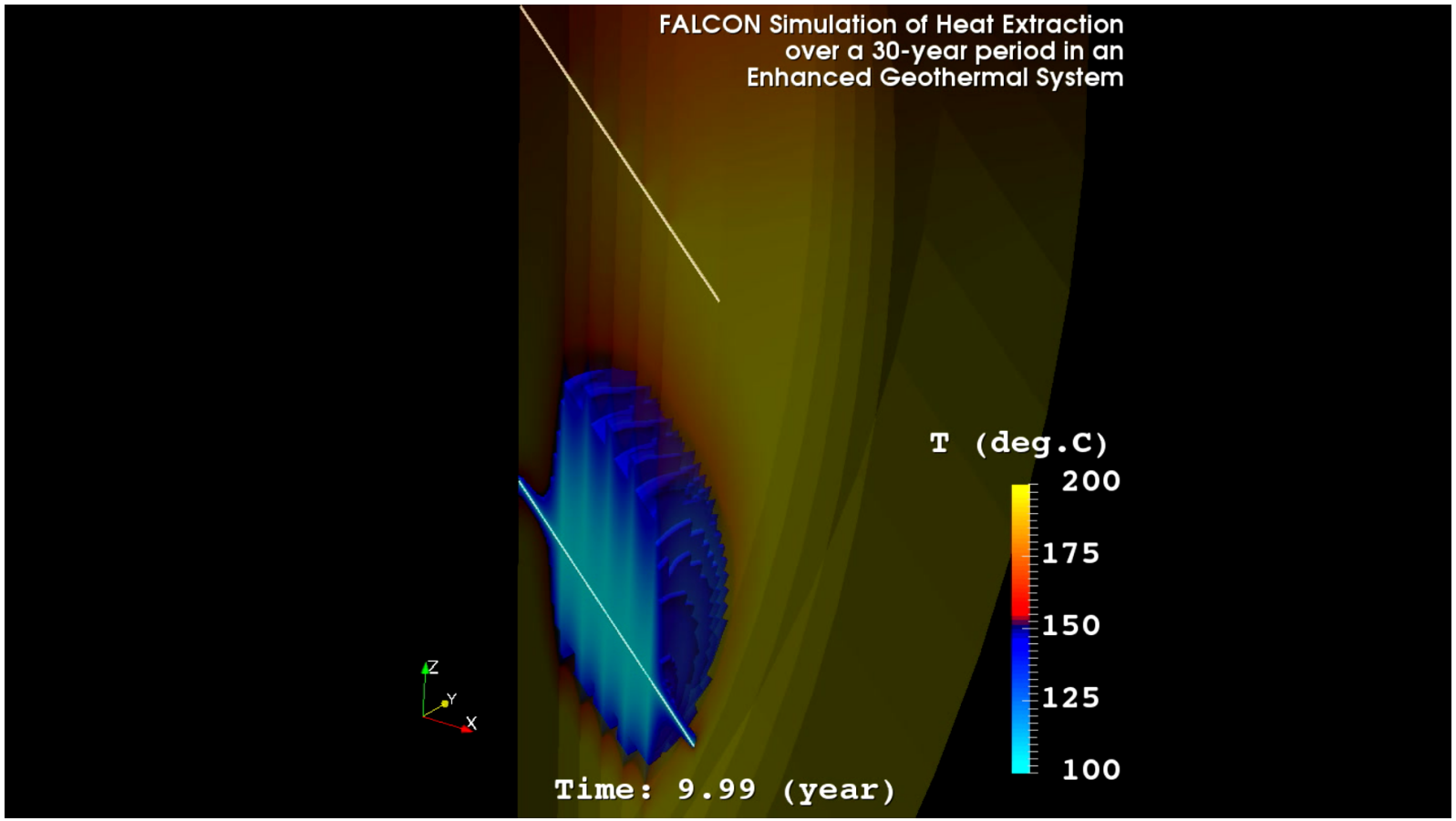
MOOSE + FALCON - Fracturing And Liquid CONservation

`https://github.com/idaholab/falcon`



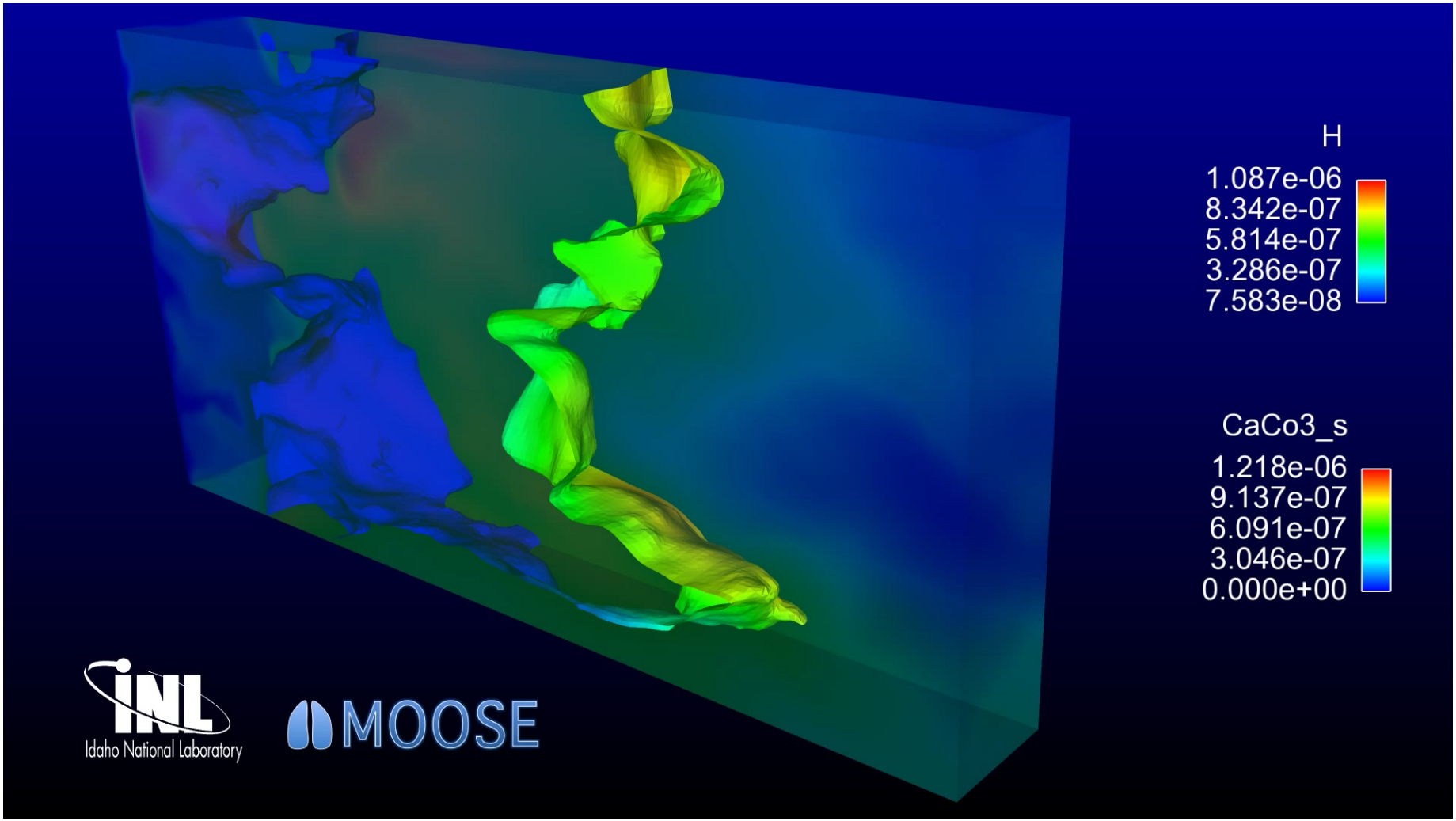
MOOSE + FALCON - Fracturing And Liquid CONservation

`https://github.com/idaholab/falcon`



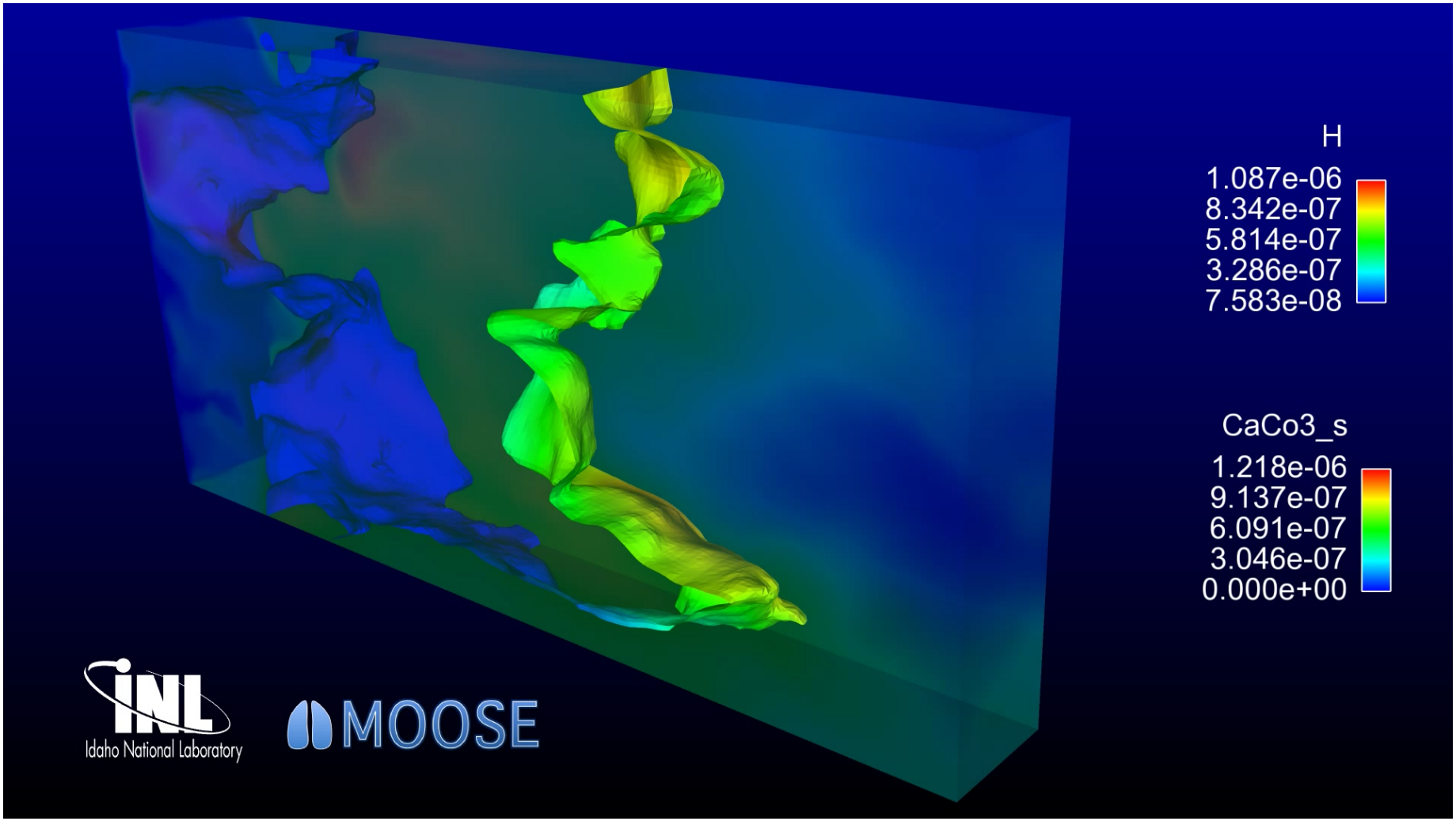
MOOSE + FALCON - Fracturing And Liquid CONservation

`https://github.com/idaholab/falcon`



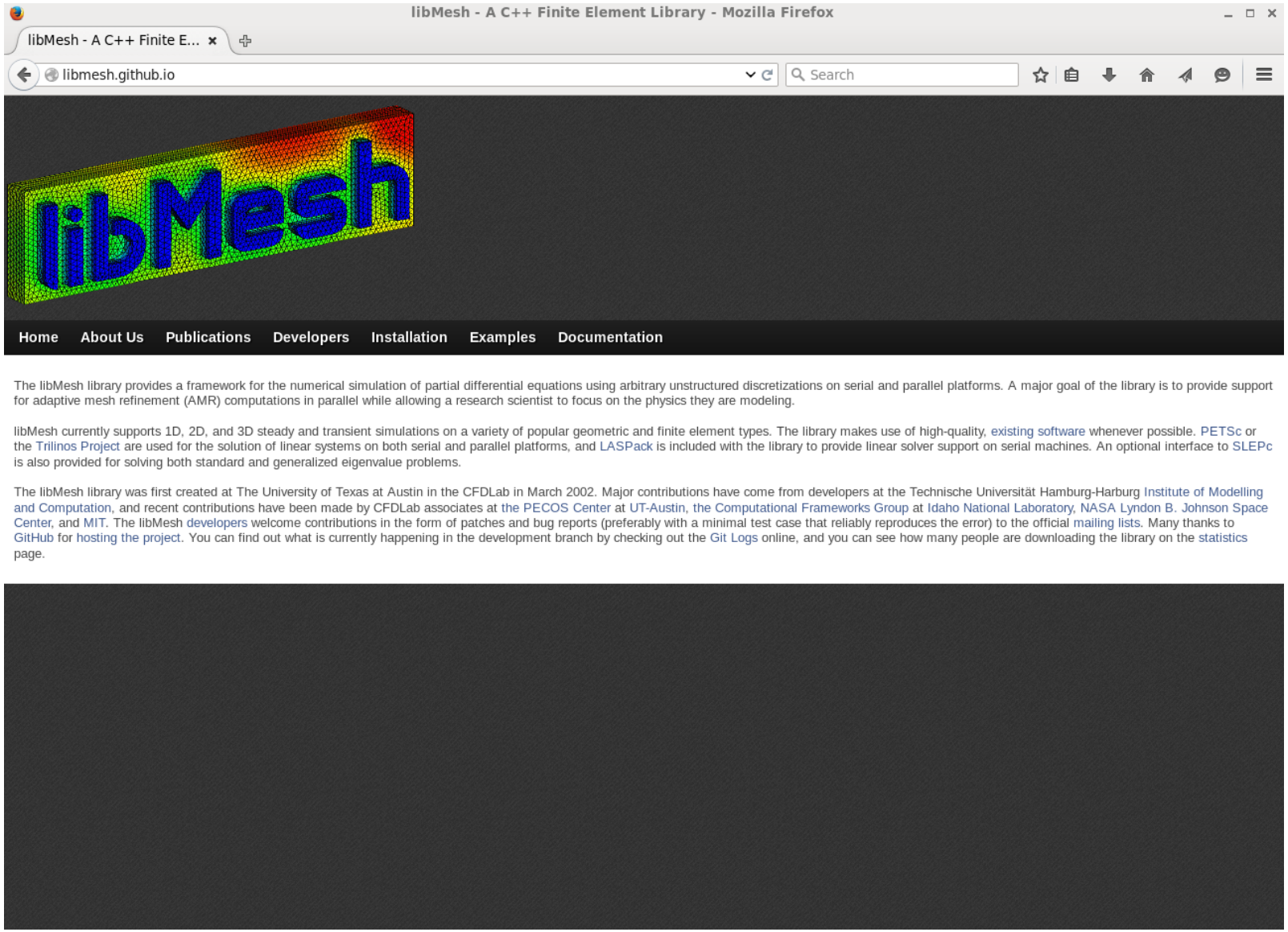
MOOSE + FALCON - Fracturing And Liquid CONservation

`https://github.com/idaholab/falcon`

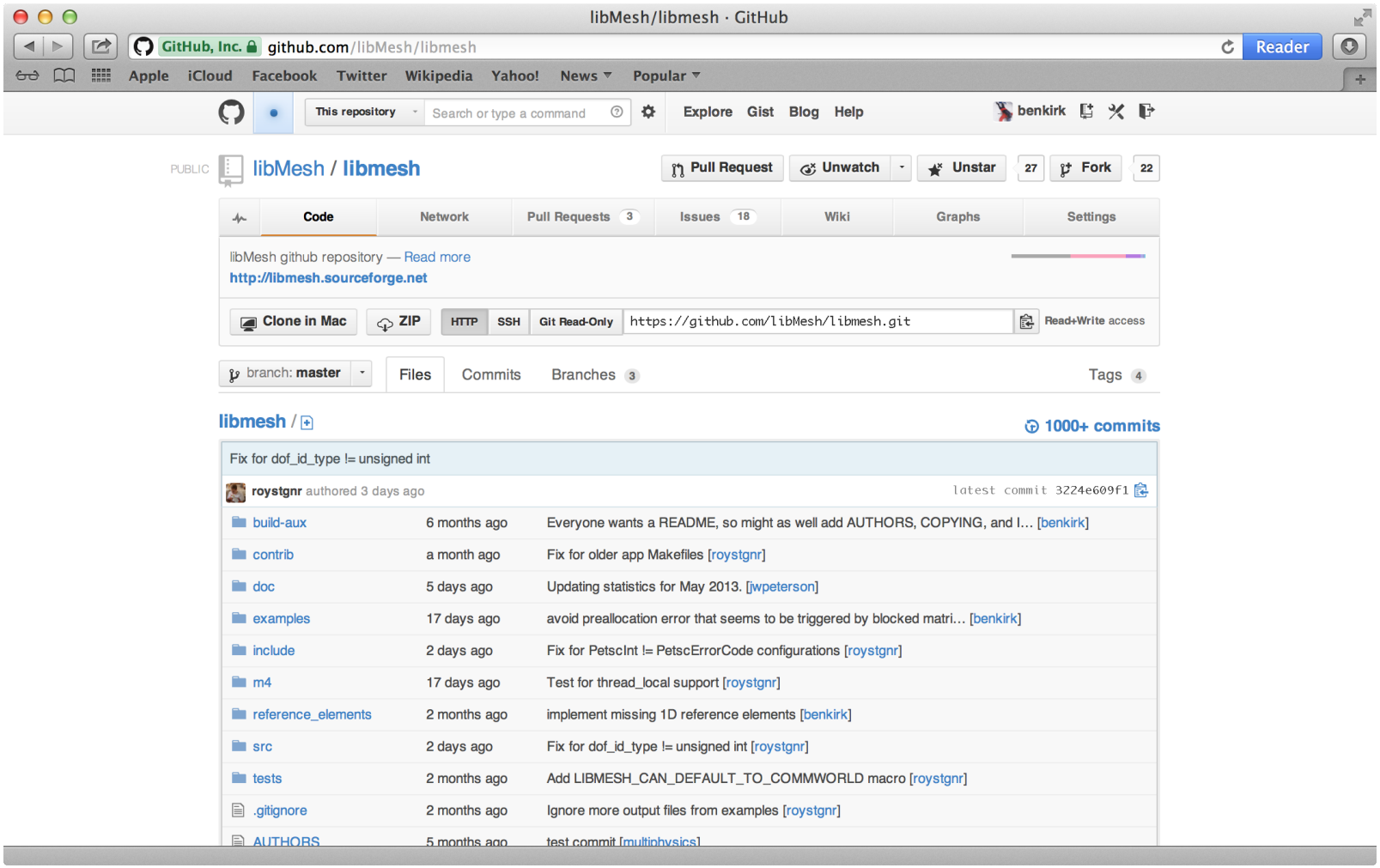


Outline

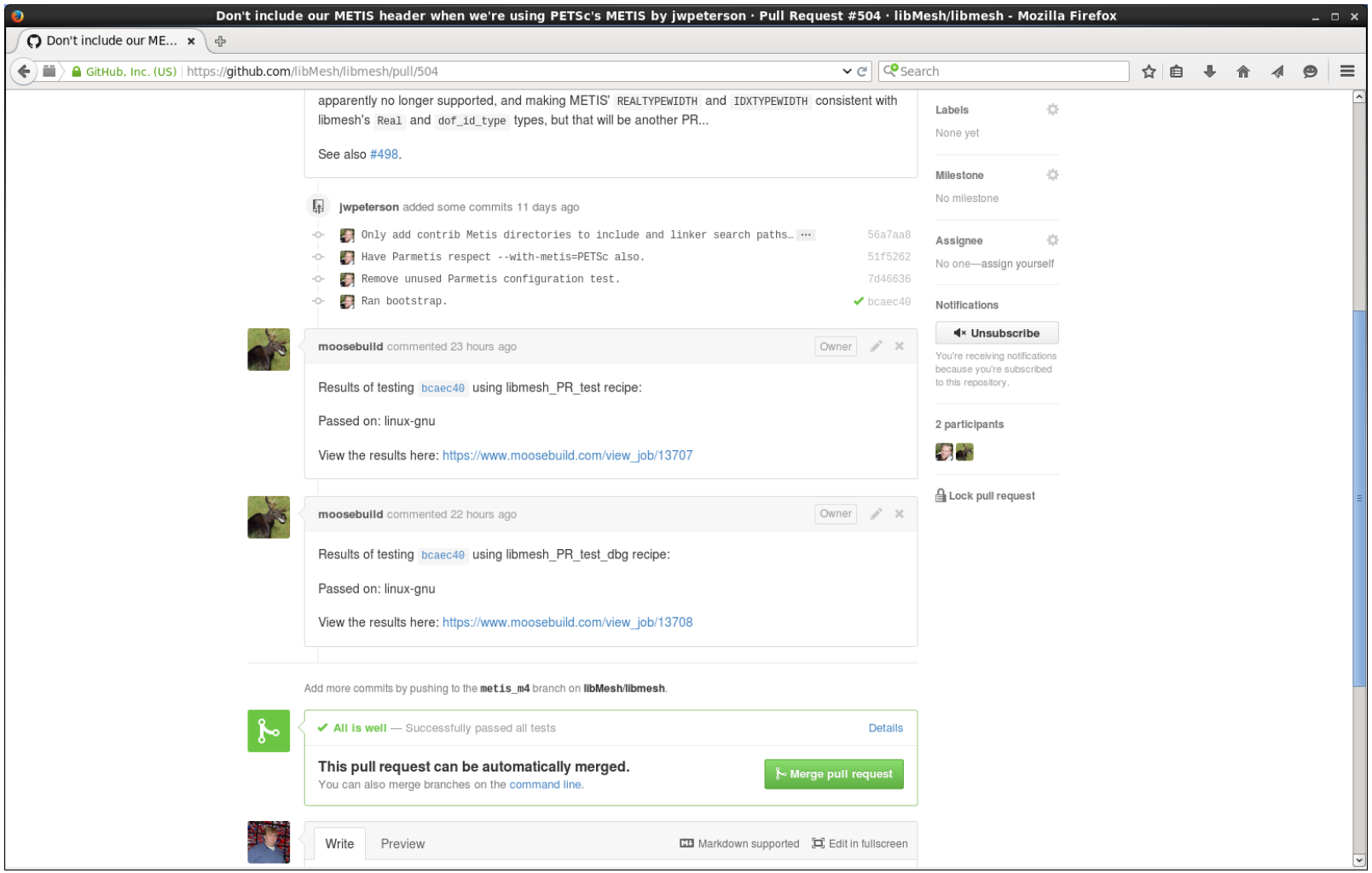
http://libmesh.github.io



http://github.com/libMesh/libmesh



Continuous Integration



Getting the libMesh Source

- **Blessed, Stable releases:**
  Download prepackaged releases from
  `http://github.com/libMesh/libmesh/releases`
- **Development tree:**
  Grab the latest source tree from GitHub:

```
$ git clone git://github.com/libMesh/libmesh.git
```

libMesh Suggested Dependencies

- MPI is of course required for distributed-memory parallelism.
- Out of the box, libMesh will build with support for serial linear systems.
- Highly recommended you first install PETSc and/or Trilinos, which libMesh uses for solving linear systems in parallel.
- Other recommended, optional packages include:
  - SLEPc: eigenvalue support on top of PETSc.
  - Intel's Threading Building Blocks for shared-memory multithreading.

Building libMesh from source

Unpack, Configure, Build, Install, & Test

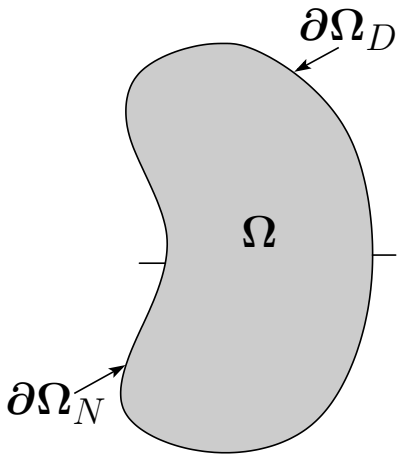
```
# unpack the distribution
$ tar jxf libmesh-0.9.5.tar.bz2 && cd libmesh-0.9.5
# configure, install into the current directory
$ ./configure --prefix=$PWD/install
# build & install
$ make -j 4 && make -j 4 install
# run all the examples, but only the optimized flavor
$ make -j 4 check METHODS=opt
```

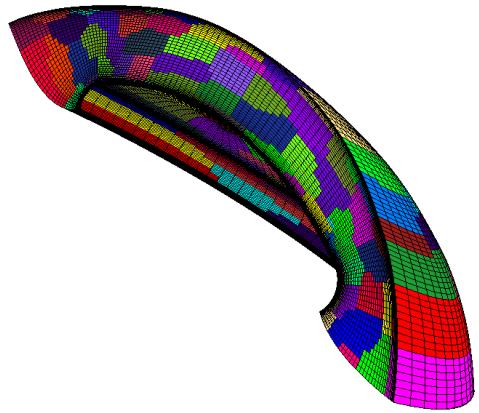
Outline

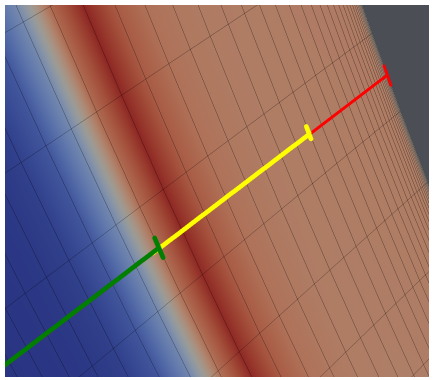
Typical Boundary Value Problem

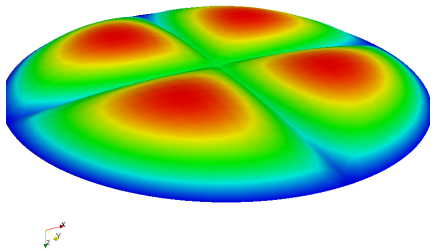
- Common BVP components:

$$
\begin{aligned}
\boldsymbol{M}\frac{\partial \boldsymbol{u}}{\partial t} &= \boldsymbol{F}(\boldsymbol{u}) && \in \Omega \subset \mathbb{R}^n \\
\boldsymbol{G}(\boldsymbol{u}) &= 0 && \in \Omega \\
\boldsymbol{u} &= \boldsymbol{u}_D && \in \partial\Omega_D \\
\boldsymbol{N}(\boldsymbol{u}) &= 0 && \in \partial\Omega_N \\
\boldsymbol{u}(\boldsymbol{x}, 0) &= u_0(\boldsymbol{x})
\end{aligned}
$$





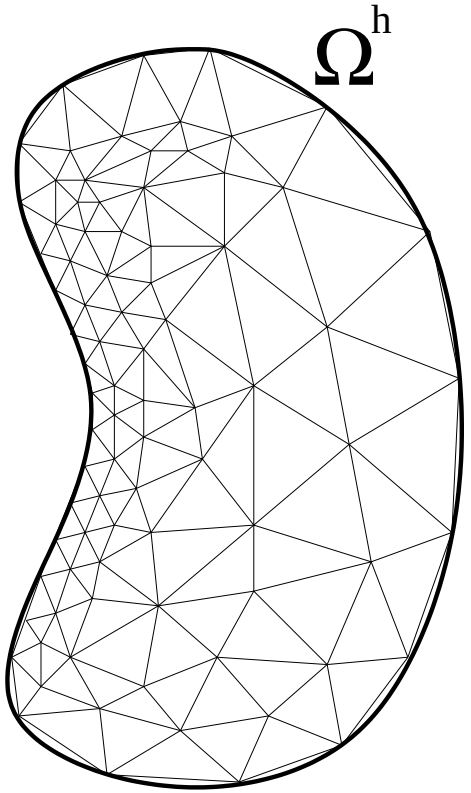




- Less common components:
  - Moving domain $\Omega(t)$, $\Omega(\boldsymbol{u}, t)$
  - Multi-dimensional manifolds
  - Self-overlapping, contact
  - Acceleration $\partial^2 u / \partial t^2$
  - Integro-differential equations

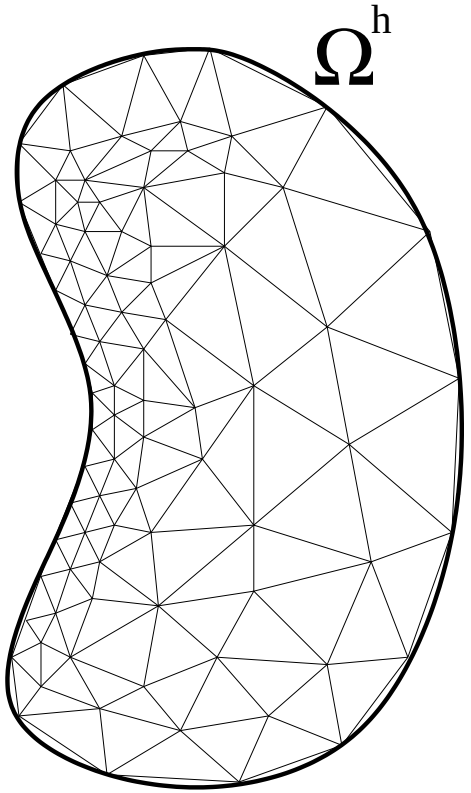
- Associated to the problem domain Ω is a `libMesh` data structure called a `Mesh`
- A `Mesh` is essentially a collection of finite elements

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



- 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, interfaces to Triangle and TetGen, and supports a rich set of input file formats.

Outline

Operations on Objects in the `Mesh`

- From a `Mesh` it is trivial to access ranges of objects of interest through *iterators*.
- Iterators are simply a mechanism for accessing a range of objects.
- `libMesh` makes extensive use of *predictated iterators* to access, for example,
  - All elements in the mesh.
  - The "active" elements in the mesh assigned to the local processor in a parallel simulation.
  - The nodes in the mesh.

Mesh Iterators

```
void foo (const MeshBase &mesh)
{
  // Now we will loop over all the elements in the mesh that
  // live on the local processor. We will compute the element
  // matrix and right-hand-side contribution.  Since the mesh
  // may be refined we want to only consider the ACTIVE elements,
  // hence we use a variant of the \p active_elem_iterator.
  MeshBase::const_element_iterator
    el     = mesh.active_local_elements_begin();
  const MeshBase::const_element_iterator
    end_el = mesh.active_local_elements_end();

  for ( ; el != end_el; ++el)
    {
      // Store a pointer to the element we are currently
      // working on.  This allows for nicer syntax later.
      const Elem* elem = *el;
      ...
    }
  ...
}
```

Mesh Iterators

```
void foo (const MeshBase &mesh)
{
  // We will now loop over all nodes.
  MeshBase::const_node_iterator       node_it  = mesh.nodes_begin();
  const MeshBase::const_node_iterator node_end = mesh.nodes_end();

  for ( ; node_it != node_end; ++node_it)
    {
      // the current node pointer
      const Node* node = *node_it;
      ...
    }
  ...
}
```

EquationSystems

- The `Mesh` is a discrete representation of the geometry for a problem.
- For a given `Mesh`, there can be an `EquationSystems` object, which represents one or more coupled system of equations posed on the `Mesh`.
  - There is only one `EquationSystems` object per `Mesh` object.
  - The `EquationSystems` object can hold many `System` objects, each representing a logical system of equations.
- High-level operations such as solution input/output is usually handled at the `EquationSystems` level.

EquationSystems

```
...
// Create an equation systems object. This object can contain
// multiple systems of different flavors for solving loosely coupled
// physics.  Each system can contain multiple variables of different
// approximation orders. The EquationSystems object needs a
// reference to the mesh object, so the order of construction here
// is important.
EquationSystems equation_systems (mesh);

// Now we declare the system and its variables.  We begin by adding
// a "TransientLinearImplicitSystem" to the EquationSystems object,
// and we give it the name "Simple System".
equation_systems.add_system<TransientLinearImplicitSystem> ("Simple System");

// Adds the variable "u" to "Simple System".  "u" will be
// approximated using first-order approximation.
equation_systems.get_system("Simple System").add_variable("u", FIRST);

// Next we'll by add an "ExplicitSystem" to the EquationSystems
// object, and we give it the name "Complex System".
equation_systems.add_system<ExplicitSystem> ("Complex System");

// Give "Complex System" three variables -- each with a different
// approximation order.  Variables "c" and "T" will use first-order
// Lagrange approximation, while variable "dv" will use a
// second-order discontinuous approximation space.
equation_systems.get_system("Complex System").add_variable("c", FIRST);
equation_systems.get_system("Complex System").add_variable("T", FIRST);
equation_systems.get_system("Complex System").add_variable("dv", SECOND, MONOMIAL);

// Initialize the data structures for the equation system.
equation_systems.init();

// Prints information about the system to the screen.
equation_systems.print_info();
...
```

Elements

- The `Elem` base class defines a geometric element in `libMesh`.
- An `Elem` is defined by `Node`s, Edges (2D,3D) and Faces (3D).
- An `Elem` is sufficiently rich that in many cases it is the only argument required to provide to a function.

Elements

```
// access each Node on the element
for (unsigned int n=0; n<elem->n_nodes(); n++)
  {
    const Node *node = elem->get_node(n);

    // get a user-specified material property, based on
    // the subdomain the element belongs to
    const Real k_diff = my_matprop_func (elem->subdomain_id(), *node);
    ...
  }


// Perform some operation for elements on the boundary
for (unsigned int side=0; side<elem->n_sides(); side++)
  {
    // Every element knows its neigbor.  If it has no neighbor,
    // then it lies on a physical boundary.
    if (elem->neighbor(side) == NULL)
      {
        // Construct the side as a lower dimensional element
        AutoPtr<Elem> elem_side (elem->build_side(side));
        ...
      }
    ...
  }
```

Nodes

- Nodes define spatial locations in arbitrary dimensions.
- Logically, a Node is a point in *N*-space plus metadata:
  - Global ID.
  - Processor ownership.
  - Degree of freedom indexing data.

Nodes

```
// loop over a range and determine the bounding box
void bounding_box(const ConstNodeRange &range)
{
  ...
  for (ConstNodeRange::const_iterator it = range.begin();
       it != range.end(); ++it)
    {
      const Node *node = *it;

      for (unsigned int i=0; i<LIBMESH_DIM; i++)
        {
          _vmin[i] = std::min(_vmin[i], (*node)(i));
          _vmax[i] = std::max(_vmax[i], (*node)(i));
        }
    }
  ...
}


...
// Query the number of DOFs for a particular node in a system
const unsigned int n_dofs_per_node = node->n_dofs(sys_num);
```

Outline

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

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

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

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

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

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

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

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

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

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

- Even stabilized formulations boil down to some semilinear form

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

Poisson Equation

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

Poisson Equation

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

Weighted Residual Statement

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

Linear Convection-Diffusion

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

Linear Convection-Diffusion

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

Weighted Residual Statement

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

Stokes Flow

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

Stokes Flow

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

Weighted Residual Statement

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

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

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

- 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.

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

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

- The integral over Ω^h ...

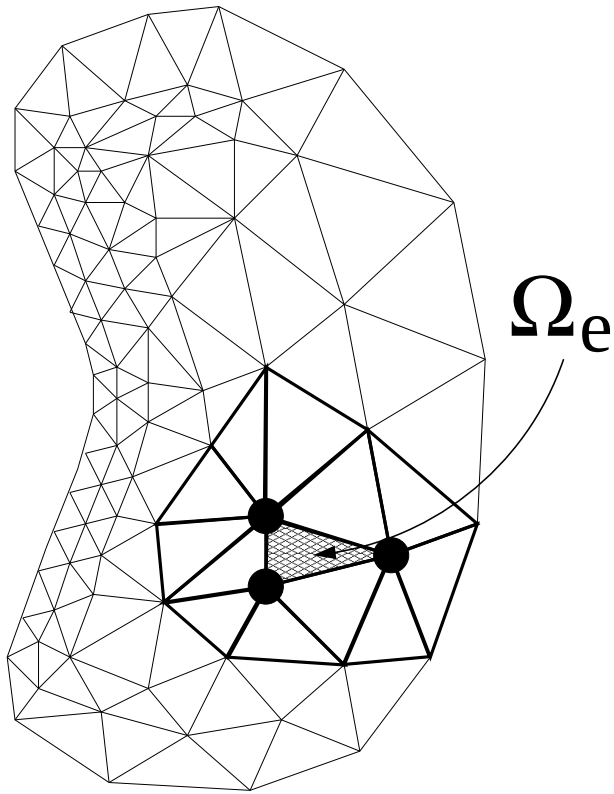
$$0 = \int_{\Omega^h} \left[\nabla u^h \cdot \nabla v^h - f v^h\right] dx \quad \forall v^h \in \mathcal{V}^h$$

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

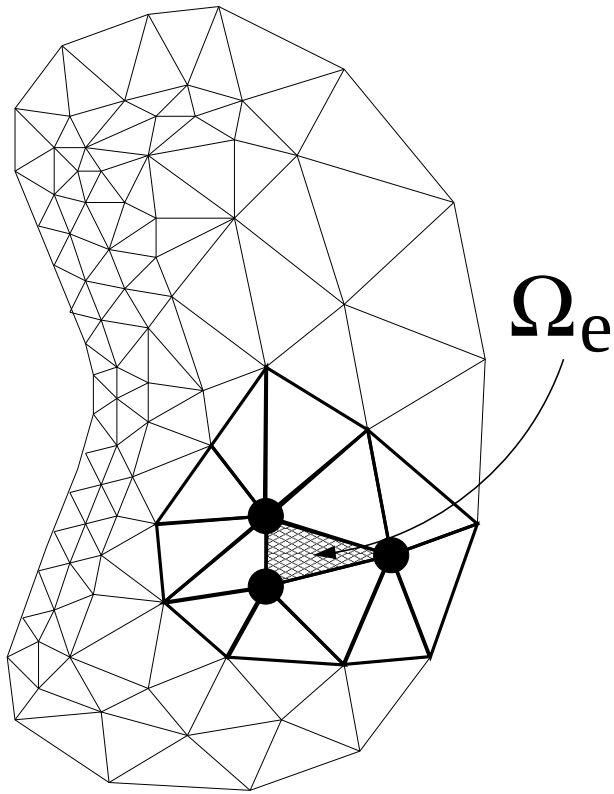
$$
\begin{aligned}
0 &= \int_{\Omega^h} \left[\nabla u^h \cdot \nabla v^h - f v^h\right] dx \quad \forall v^h \in \mathcal{V}^h \\
&= \sum_{e=1}^{N_e} \int_{\Omega_e} \left[\nabla u^h \cdot \nabla v^h - f v^h\right] dx \quad \forall v^h \in \mathcal{V}^h
\end{aligned}
$$

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

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



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



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

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

- 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

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

- The entries of the element stiffness matrix are the integrals

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

- 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 entries of the element stiffness matrix are the integrals

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

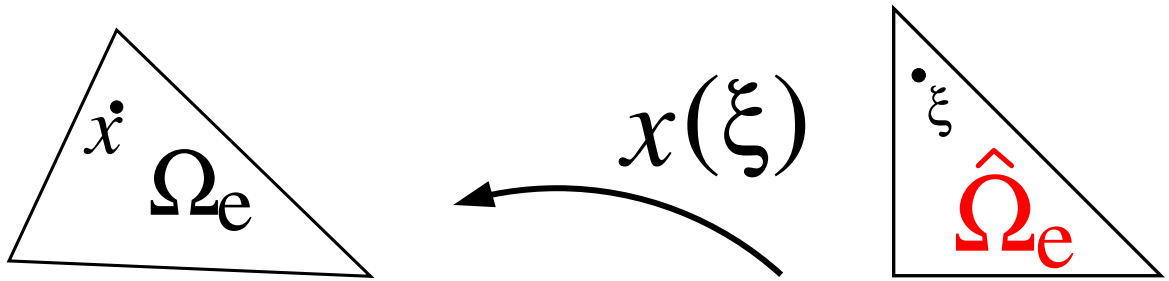
- While for the element right-hand side we have

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

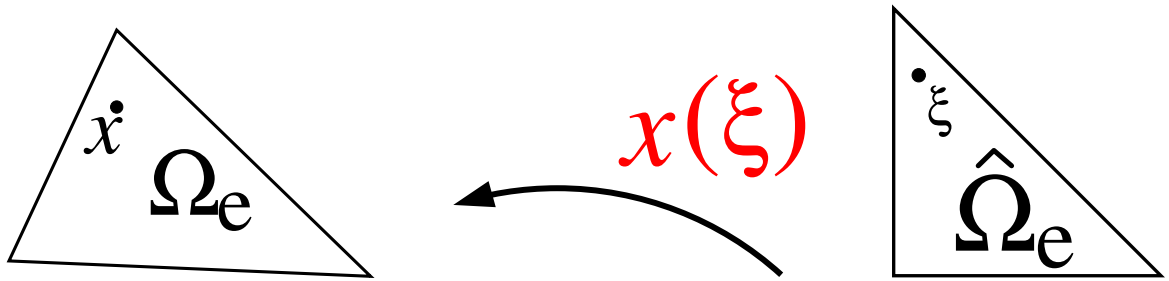
- The element stiffness matrices and right-hand sides can be "assembled" to obtain the global system of equations

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

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



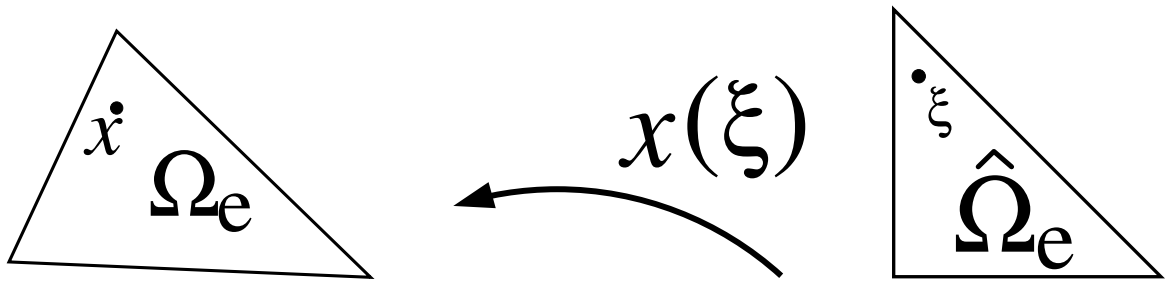
- The integrals are performed on a “reference” element $\hat{\Omega}_e$



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

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

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



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

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

- 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".

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

- 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]	$\|J(\xi_q)\|w_q$	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
d2phi[i][q]	$\hat{\nabla}^2_\xi \phi_i(\xi_q)$	value of i^{th} shape fn. Hessian
xyz[q]	$x(\xi_q)$	location of ξ_q in physical space

- 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]);
  }
```

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

- 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]);
  }
```

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

- 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]);
  }
```

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

- 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 matrix assembly routine for the linear convection-diffusion equation,

$$-k\Delta u + \boldsymbol{b} \cdot \nabla u = f$$

```
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]*(k*(dphi[j][q]*dphi[i][q])
                        +(b*dphi[j][q])*phi[i][q]);
  }
```

- The matrix assembly routine for the linear convection-diffusion equation,

$$-k\Delta u + \boldsymbol{b} \cdot \nabla u = f$$

```
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]*(k*(dphi[j][q]*dphi[i][q])
                        +(b*dphi[j][q])*phi[i][q]);
  }
```

- The matrix assembly routine for the linear convection-diffusion equation,

$$-k\Delta u + \boldsymbol{b} \cdot \nabla u = f$$

```
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]*(k*(dphi[j][q]*dphi[i][q])
                        +(b*dphi[j][q])*phi[i][q]);
  }
```

- For multi-variable systems like Stokes flow,

$$\begin{aligned} \nabla p - \nu \Delta \boldsymbol{u} &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= 0 \end{aligned} \qquad \in \quad \Omega \subset \mathbb{R}^2$$

- The element stiffness matrix concept can extended to include sub-matrices

$$\left[\begin{array}{cc|c} K^e_{u_1u_1} & K^e_{u_1u_2} & K^e_{u_1p} \\ K^e_{u_2u_1} & K^e_{u_2u_2} & K^e_{u_2p} \\ \hline K^e_{pu_1} & K^e_{pu_2} & K^e_{pp} \end{array}\right] \left[\begin{array}{c} U^e_{u_1} \\ U^e_{u_2} \\ \hline U^e_p \end{array}\right] - \left[\begin{array}{c} F^e_{u_1} \\ F^e_{u_2} \\ \hline F^e_p \end{array}\right]$$

- We have an array of submatrices: `Ke[ ][ ]`

- For multi-variable systems like Stokes flow,

$$\begin{aligned} \nabla p - \nu \Delta \boldsymbol{u} &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= 0 \end{aligned} \qquad \in \quad \Omega \subset \mathbb{R}^2$$

- The element stiffness matrix concept can extended to include sub-matrices

$$\left[\begin{array}{cc|c} K^e_{u_1u_1} & K^e_{u_1u_2} & K^e_{u_1p} \\ K^e_{u_2u_1} & K^e_{u_2u_2} & K^e_{u_2p} \\ \hline K^e_{pu_1} & K^e_{pu_2} & K^e_{pp} \end{array}\right] \left[\begin{array}{c} U^e_{u_1} \\ U^e_{u_2} \\ \hline U^e_p \end{array}\right] - \left[\begin{array}{c} F^e_{u_1} \\ F^e_{u_2} \\ \hline F^e_p \end{array}\right]$$

- We have an array of submatrices: `Ke[0][0]`

- For multi-variable systems like Stokes flow,

$$\begin{aligned} \nabla p - \nu \Delta \boldsymbol{u} &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= 0 \end{aligned} \qquad \in \quad \Omega \subset \mathbb{R}^2$$

- The element stiffness matrix concept can extended to include sub-matrices

$$\left[\begin{array}{cc|c} K^e_{u_1u_1} & K^e_{u_1u_2} & K^e_{u_1p} \\ K^e_{u_2u_1} & K^e_{u_2u_2} & K^e_{u_2p} \\ \hline K^e_{pu_1} & K^e_{pu_2} & K^e_{pp} \end{array}\right] \left[\begin{array}{c} U^e_{u_1} \\ U^e_{u_2} \\ \hline U^e_p \end{array}\right] - \left[\begin{array}{c} F^e_{u_1} \\ F^e_{u_2} \\ \hline F^e_p \end{array}\right]$$

- We have an array of submatrices: `Ke[1][1]`

- For multi-variable systems like Stokes flow,

$$\begin{aligned} \nabla p - \nu \Delta \boldsymbol{u} &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= 0 \end{aligned} \qquad \in \quad \Omega \subset \mathbb{R}^2$$

- The element stiffness matrix concept can extended to include sub-matrices

$$\left[\begin{array}{cc|c} K^e_{u_1u_1} & K^e_{u_1u_2} & K^e_{u_1p} \\ K^e_{u_2u_1} & K^e_{u_2u_2} & K^e_{u_2p} \\ \hline K^e_{pu_1} & K^e_{pu_2} & K^e_{pp} \end{array}\right] \left[\begin{array}{c} U^e_{u_1} \\ U^e_{u_2} \\ \hline U^e_p \end{array}\right] - \left[\begin{array}{c} F^e_{u_1} \\ F^e_{u_2} \\ \hline F^e_p \end{array}\right]$$

- We have an array of submatrices: `Ke[2][1]`

- For multi-variable systems like Stokes flow,

$$\begin{aligned} \nabla p - \nu \Delta \boldsymbol{u} &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= 0 \end{aligned} \qquad \in \quad \Omega \subset \mathbb{R}^2$$

- The element stiffness matrix concept can extended to include sub-matrices

$$\left[\begin{array}{cc|c} K^e_{u_1u_1} & K^e_{u_1u_2} & K^e_{u_1p} \\ K^e_{u_2u_1} & K^e_{u_2u_2} & K^e_{u_2p} \\ \hline K^e_{pu_1} & K^e_{pu_2} & K^e_{pp} \end{array}\right] \left[\begin{array}{c} U^e_{u_1} \\ U^e_{u_2} \\ \hline U^e_p \end{array}\right] - \left[\begin{array}{c} F^e_{u_1} \\ F^e_{u_2} \\ \hline F^e_p \end{array}\right]$$

- And an array of right-hand sides: `Fe[]`.

- For multi-variable systems like Stokes flow,

$$\begin{aligned} \nabla p - \nu \Delta \boldsymbol{u} &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= 0 \end{aligned} \qquad \in \quad \Omega \subset \mathbb{R}^2$$

- The element stiffness matrix concept can extended to include sub-matrices

$$\left[\begin{array}{cc|c} K^e_{u_1u_1} & K^e_{u_1u_2} & K^e_{u_1p} \\ K^e_{u_2u_1} & K^e_{u_2u_2} & K^e_{u_2p} \\ \hline K^e_{pu_1} & K^e_{pu_2} & K^e_{pp} \end{array}\right] \left[\begin{array}{c} U^e_{u_1} \\ U^e_{u_2} \\ \hline U^e_p \end{array}\right] - \left[\begin{array}{c} F^e_{u_1} \\ F^e_{u_2} \\ \hline F^e_p \end{array}\right]$$

- And an array of right-hand sides: `Fe[0]`.

- For multi-variable systems like Stokes flow,

$$\begin{aligned} \nabla p - \nu \Delta \boldsymbol{u} &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= 0 \end{aligned} \qquad \in \quad \Omega \subset \mathbb{R}^2$$

- The element stiffness matrix concept can extended to include sub-matrices

$$\left[\begin{array}{cc|c} K^e_{u_1u_1} & K^e_{u_1u_2} & K^e_{u_1p} \\ K^e_{u_2u_1} & K^e_{u_2u_2} & K^e_{u_2p} \\ \hline K^e_{pu_1} & K^e_{pu_2} & K^e_{pp} \end{array}\right] \left[\begin{array}{c} U^e_{u_1} \\ U^e_{u_2} \\ \hline U^e_p \end{array}\right] - \left[\begin{array}{c} F^e_{u_1} \\ F^e_{u_2} \\ \hline F^e_p \end{array}\right]$$

- And an array of right-hand sides: `Fe[1]`.

- The matrix assembly can proceed in essentially the same way.
- For the momentum equations:

```
for (q=0; q<Nq; ++q)
  for (d=0; d<2; ++d)
    for (i=0; i<Ns; ++i) {
      Fe[d](i) += JxW[q]*f(xyz[q],d)*phi[i][q];

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

Outline

Model Problem

- Consider the 1D model ODE

$$\begin{cases} -u'' + bu' + cu = f \in \Omega = (0, L) \\ u(0) = u_0 \\ u(L) = u_L \end{cases} \tag{1}$$

- with weak form

$$\int_\Omega \left(u'v' + bu'v + cuv\right)\, dx = \int_\Omega fv\, dx \tag{2}$$

for every $v \in H_0^1(\Omega)$.

Model Problem (cont.)

- The analogous d-dimensional problem with $\Omega \subset \mathbb{R}^d$ and boundary $\partial\Omega$ is

$$\begin{cases} -\Delta u + \boldsymbol{b} \cdot \nabla u + cu &= f &\in \Omega \\ u &= g &\in \partial\Omega \end{cases} \tag{3}$$

- with weak form

$$\int_\Omega (\nabla u \cdot \nabla v + (\boldsymbol{b} \cdot \nabla u)v + cuv)\ dx = \int_\Omega fv\ dx \tag{4}$$

Model Problem (cont.)

- The finite element method works with the weak form, replacing the trial and test functions u, v with their approximations u^h, v^h, and summing the contributions of the element integrals

$$\sum_{e=1}^{N_e} \int_{\Omega_e} \left(\nabla u^h \cdot \nabla v^h + (\boldsymbol{b} \cdot \nabla u^h) v^h + c u^h v^h - f v^h \right) \, dx = 0 \qquad (5)$$

- Remark: We considered here a standard piecewise continuous finite element basis. In general, ∇u^h will have a jump discontinuity across element boundaries.

Galerkin FE Method

- Expressing u^h and v^h in our chosen piecewise continuous polynomial basis

$$u^h = \sum_{j=1}^{N} u_j \varphi_j \qquad v^h = \sum_{i=1}^{N} c_i \varphi_i \tag{6}$$

  we obtain on each element Ω_e

$$\sum_{j=1}^{N} u_j \left[\int_{\Omega_e} (\nabla \varphi_j \cdot \nabla \varphi_i + (\boldsymbol{b} \cdot \nabla \varphi_j)\varphi_i + c\varphi_j \varphi_i)\, dx \right] = \int_{\Omega_e} f \varphi_i \, dx \tag{7}$$

  for $i = 1 \ldots N$.

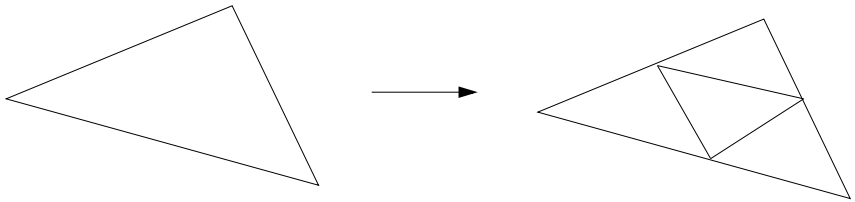
- In the standard element-stiffness matrix form,

$$\boldsymbol{K_e U} = \boldsymbol{F_e} \tag{8}$$

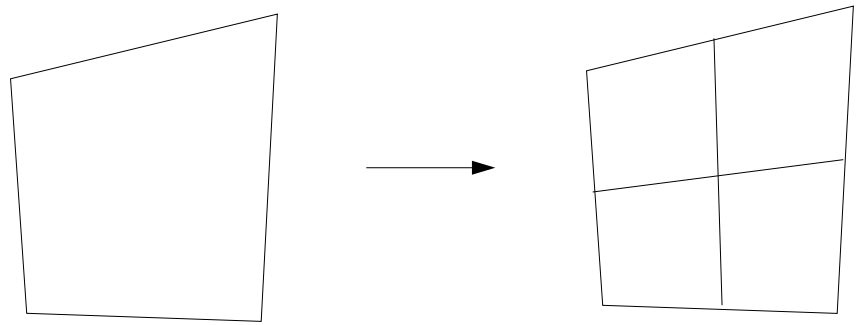
LibMesh Representation (cont.)

```
for (q=0; q<Nq; ++q) {
  // Compute b, c, f at this quadrature point
  // ...

  for (i=0; i<N; ++i) {
    Fe(i)   += JxW[q]*f*phi[i][q];

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

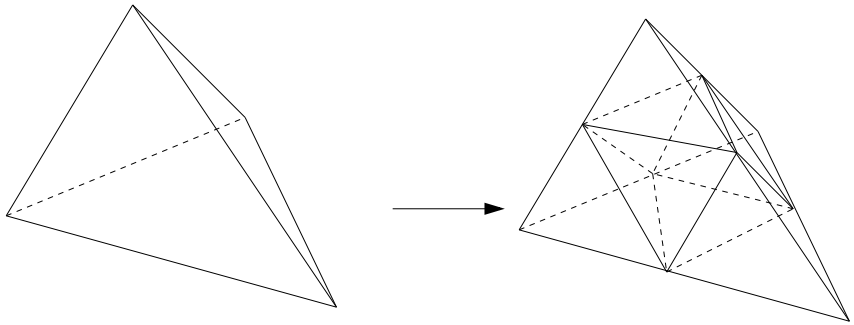
Natural Refinement Patterns



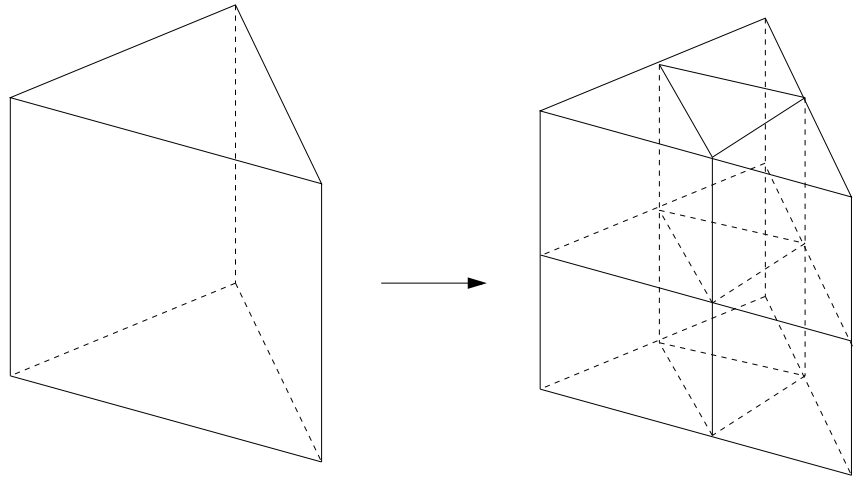
Triangle



Quadrilateral



Tetrahedron



Prism

Flux-Jump Error Indicator

- The flux-jump error indicator is derived starting from the element integrals

$$\sum_{e=1}^{N_e} \int_{\Omega_e} \left(\nabla u^h \cdot \nabla v^h + (\boldsymbol{b} \cdot \nabla u^h) v^h + c u^h v^h - f v^h \right) \, dx = 0 \tag{5}$$

- Applying the divergence theorem "in reverse" obtains

$$\sum_{e=1}^{N_e} \int_{\Omega_e} \left(-\Delta u^h + (\boldsymbol{b} \cdot \nabla u^h) + c u^h - f \right) v^h \, dx + \sum_{\partial\Omega_e \not\subset \partial\Omega} \int_{\partial\Omega_e} \left[\!\left[\frac{\partial u^h}{\partial n} \right]\!\right] v^h \, dx = 0 \tag{9}$$

Flux-Jump Error Indicator (cont.)

- Defining the cell residual

$$r(u^h) = -\Delta u^h + (\boldsymbol{b} \cdot \nabla u^h) + c u^h - f \tag{10}$$

we have

$$\sum_{e=1}^{N_e} \int_{\Omega_e} r(u^h) v^h \, dx + \sum_{\partial\Omega_e \not\subset \partial\Omega} \int_{\partial\Omega_e} \left[\!\left[\frac{\partial u^h}{\partial n} \right]\!\right] v^h \, dx = 0 \tag{11}$$

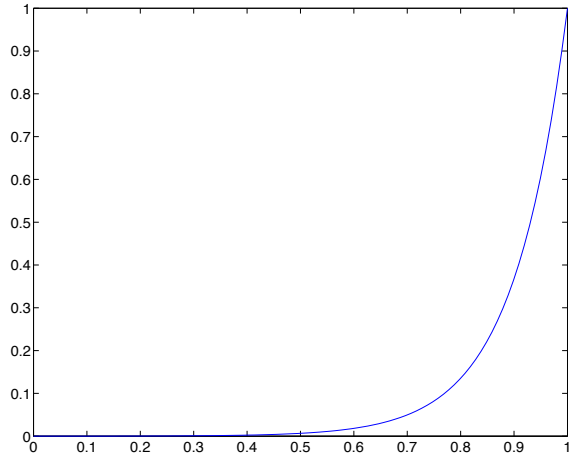
- Clearly, the exact solution u satisfies (11) identically.
- Computing $r(u^h)$ requires knowledge of the differential operator (i.e. knowledge of the "physics").
- The second sum leads to a *physics-independent* method for estimating the error in the approximate solution u^h.

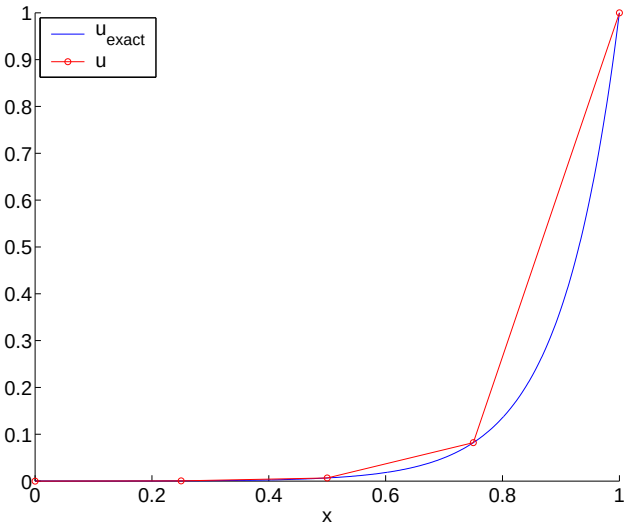
1D Example

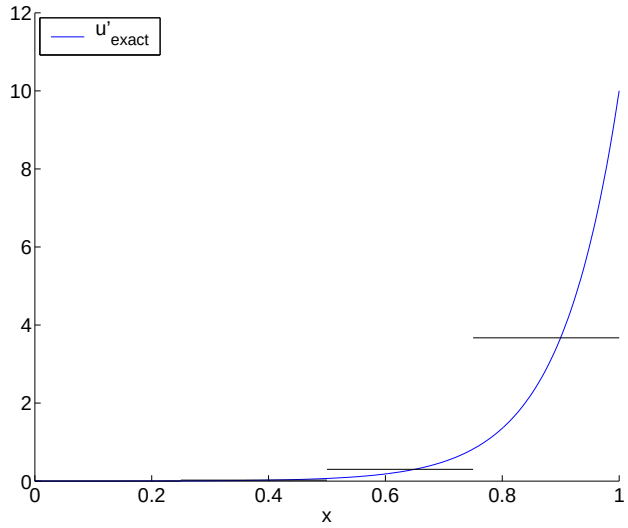
- Consider the function

$$u = \frac{1 - \exp(10x)}{1 - \exp(10)}$$

  which is a solution of the classic 1D advection-diffusion boundary layer equation.
- We assume here that the finite element solution is the linear interpolant of u, and compute the error indicator for a sequence of uniformly refined grids.

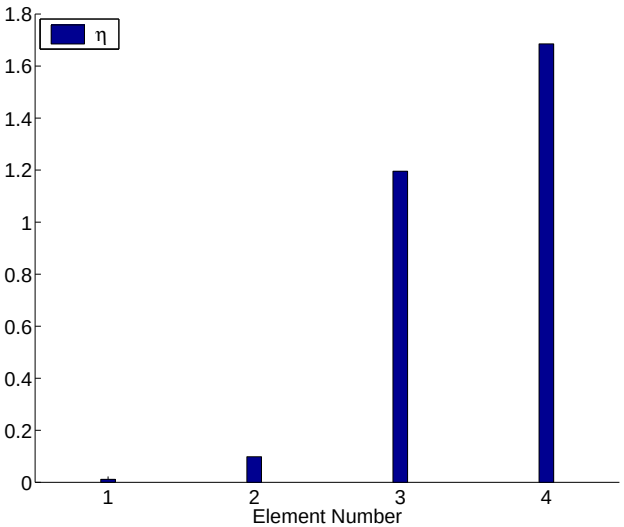


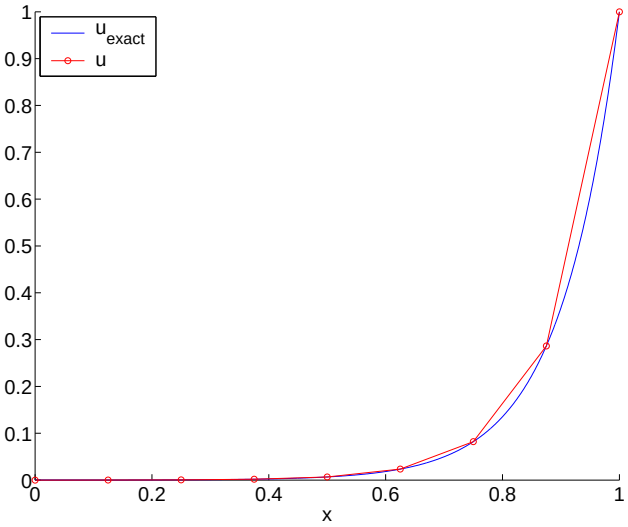


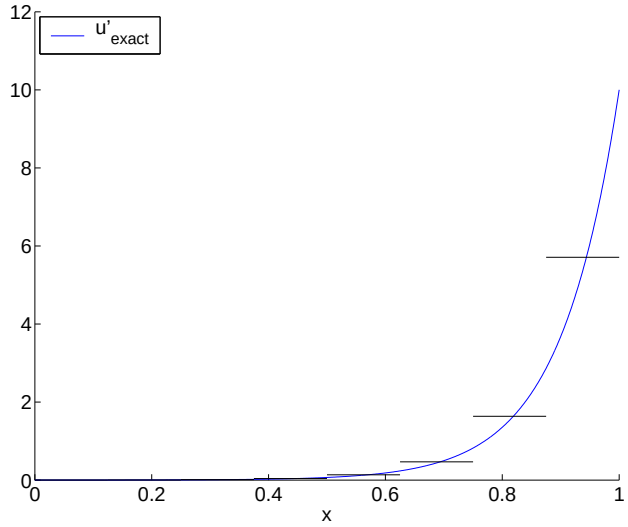


4 elements

$$||e||_{L_2} = 0.09$$

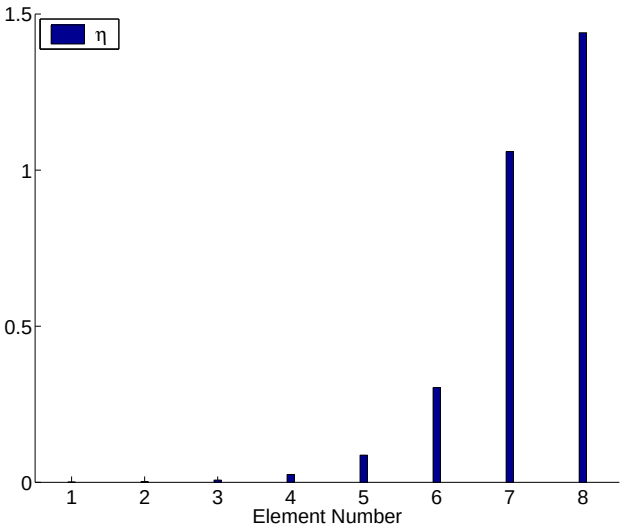


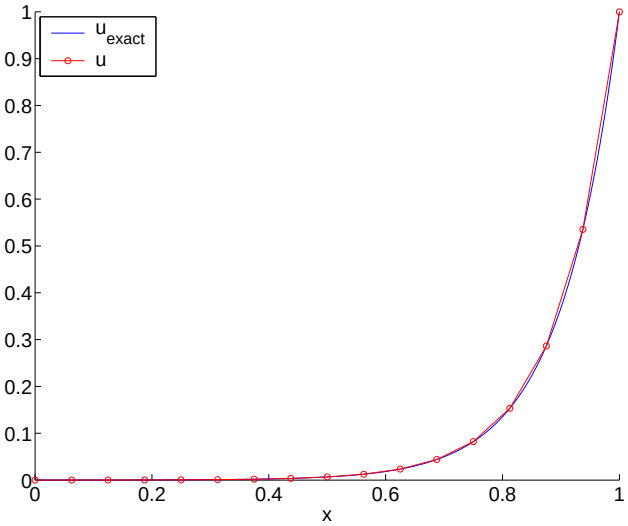


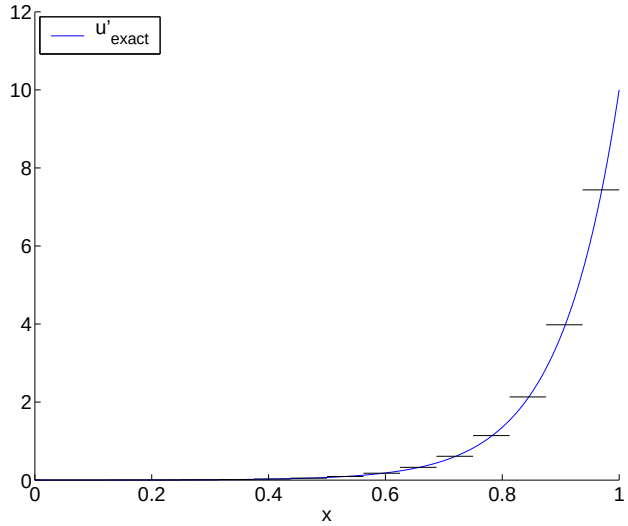


8 elements

$$||e||_{L_2} = 0.027$$

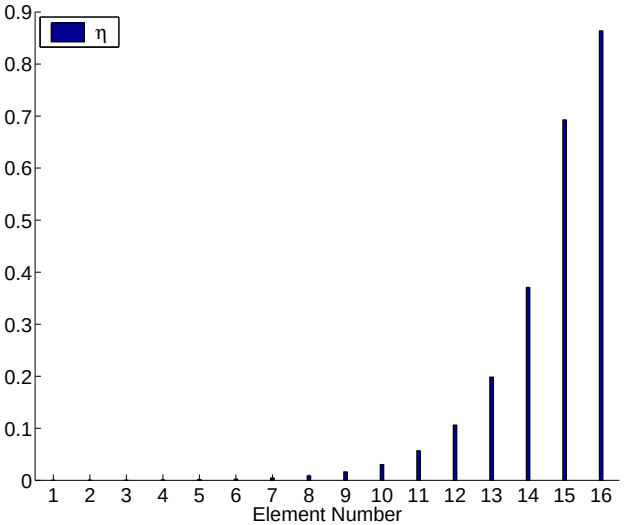






16 elements

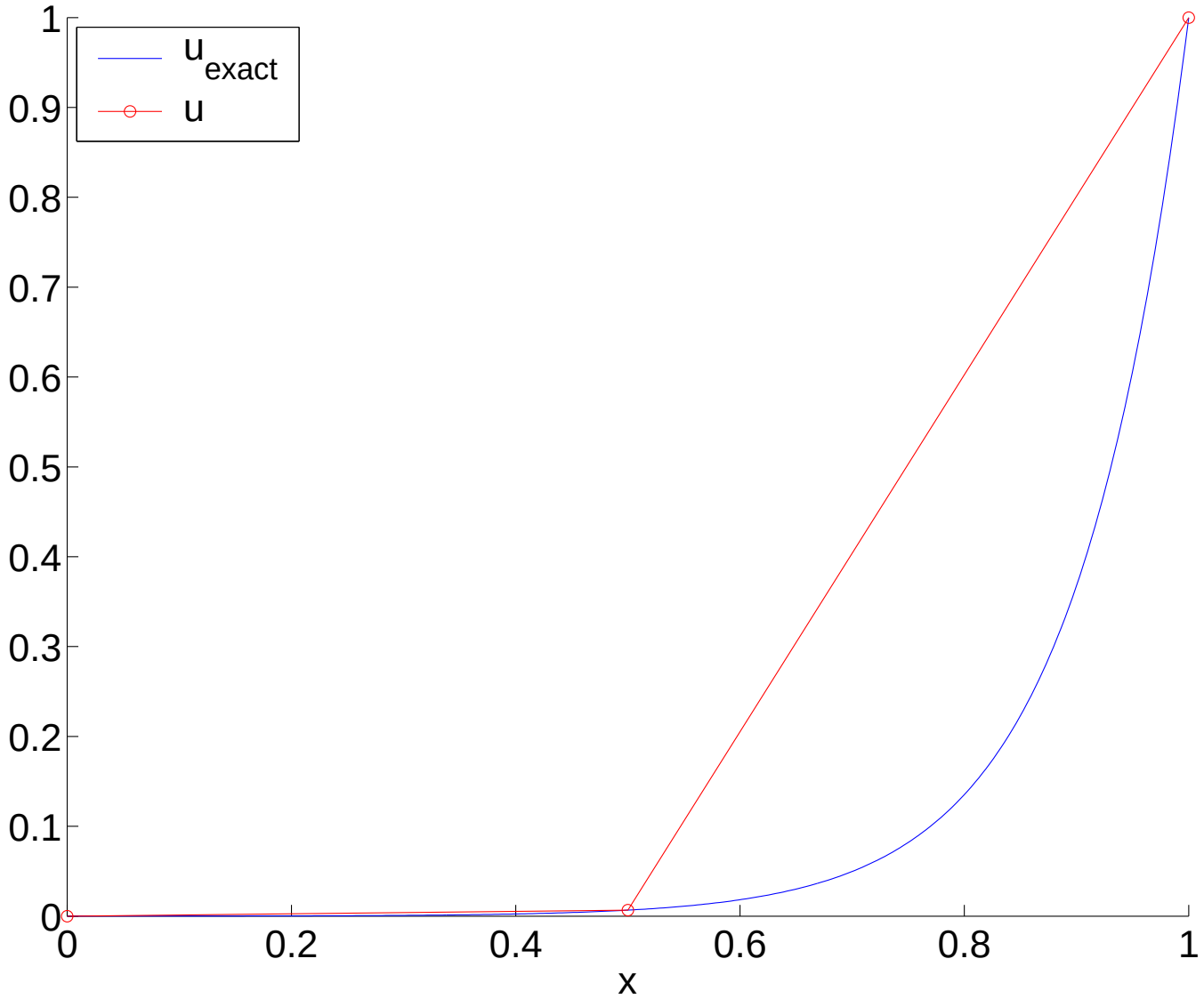
$$||e||_{L_2} = 0.0071$$

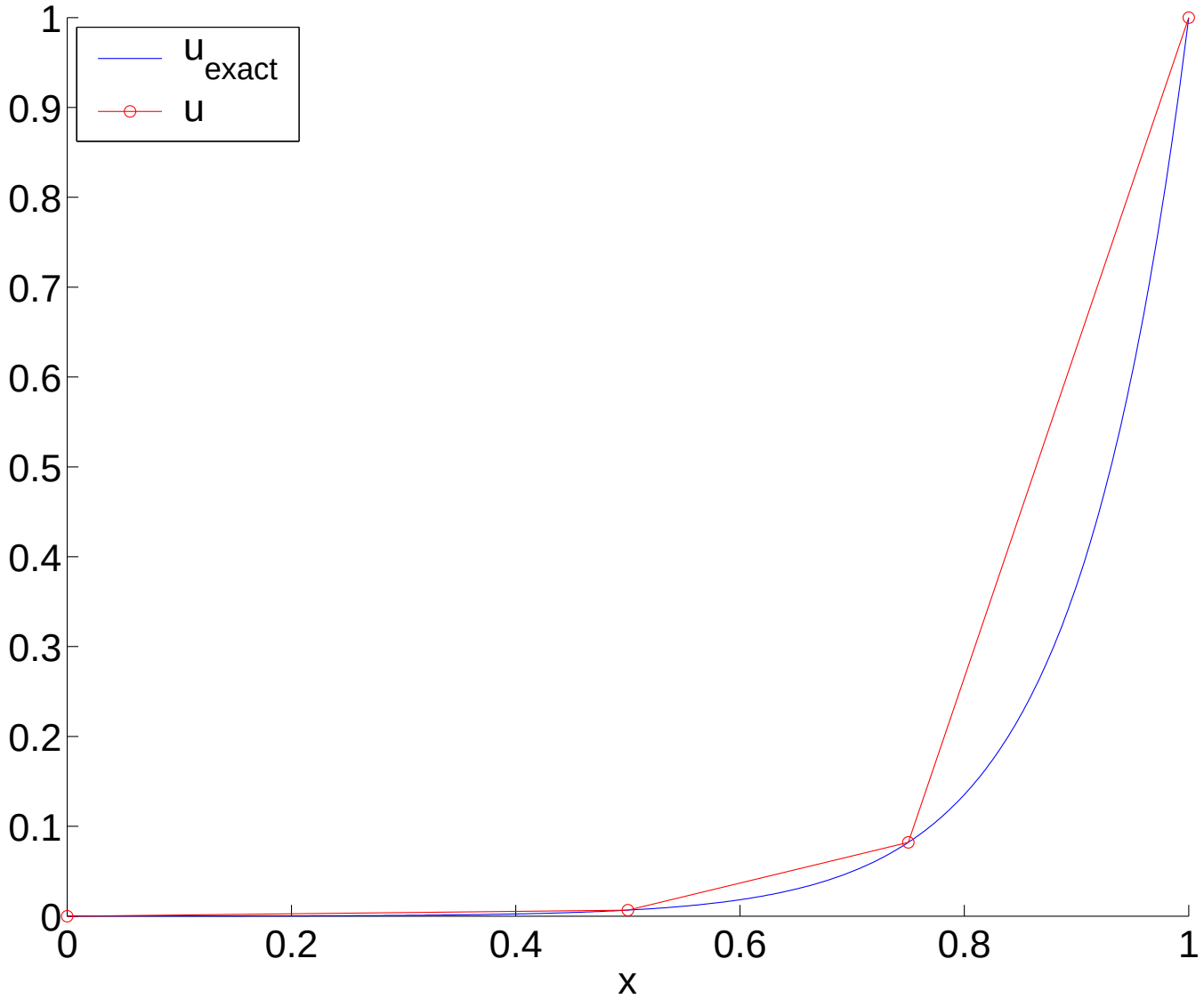


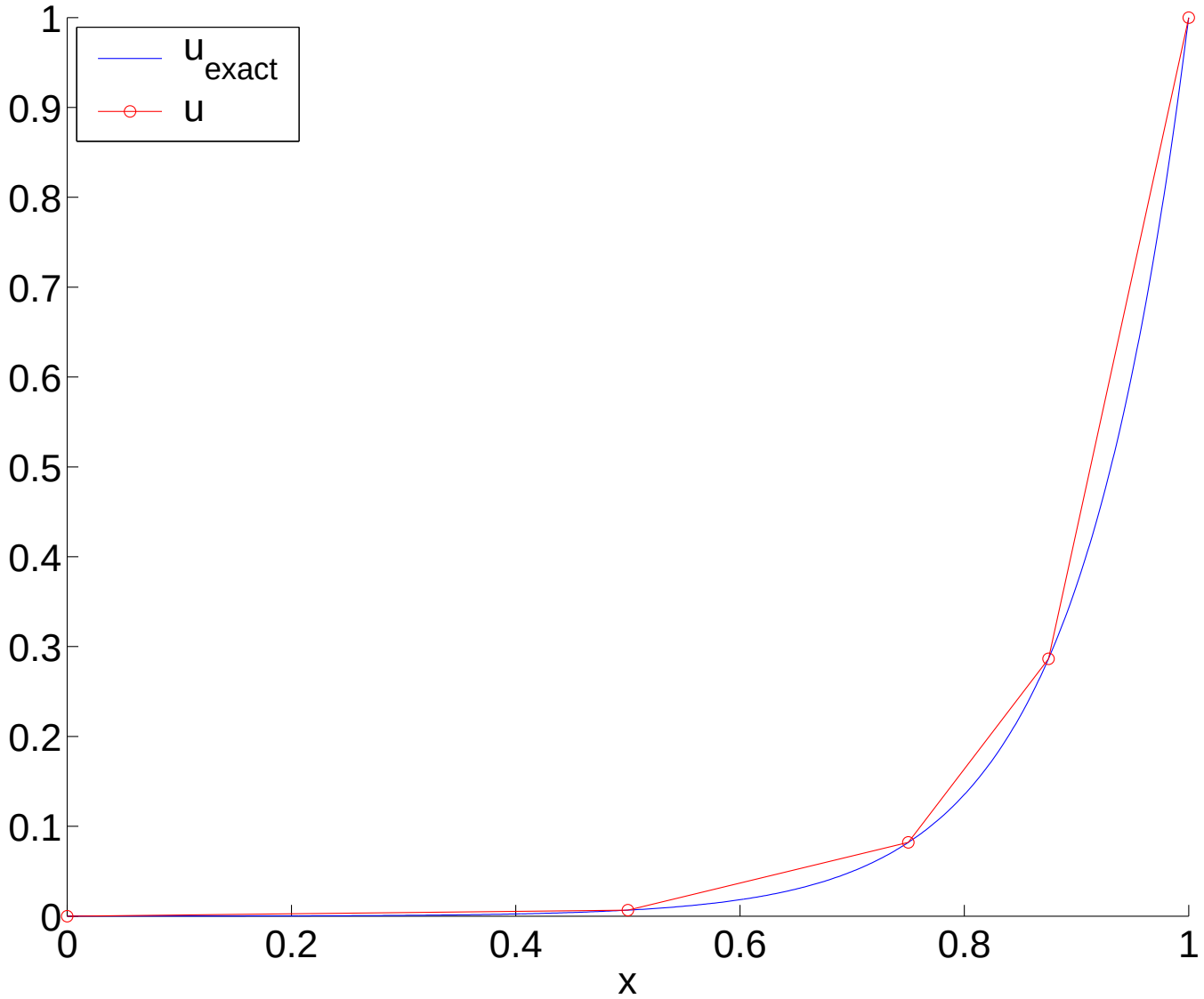
A Simple Refinement Strategy

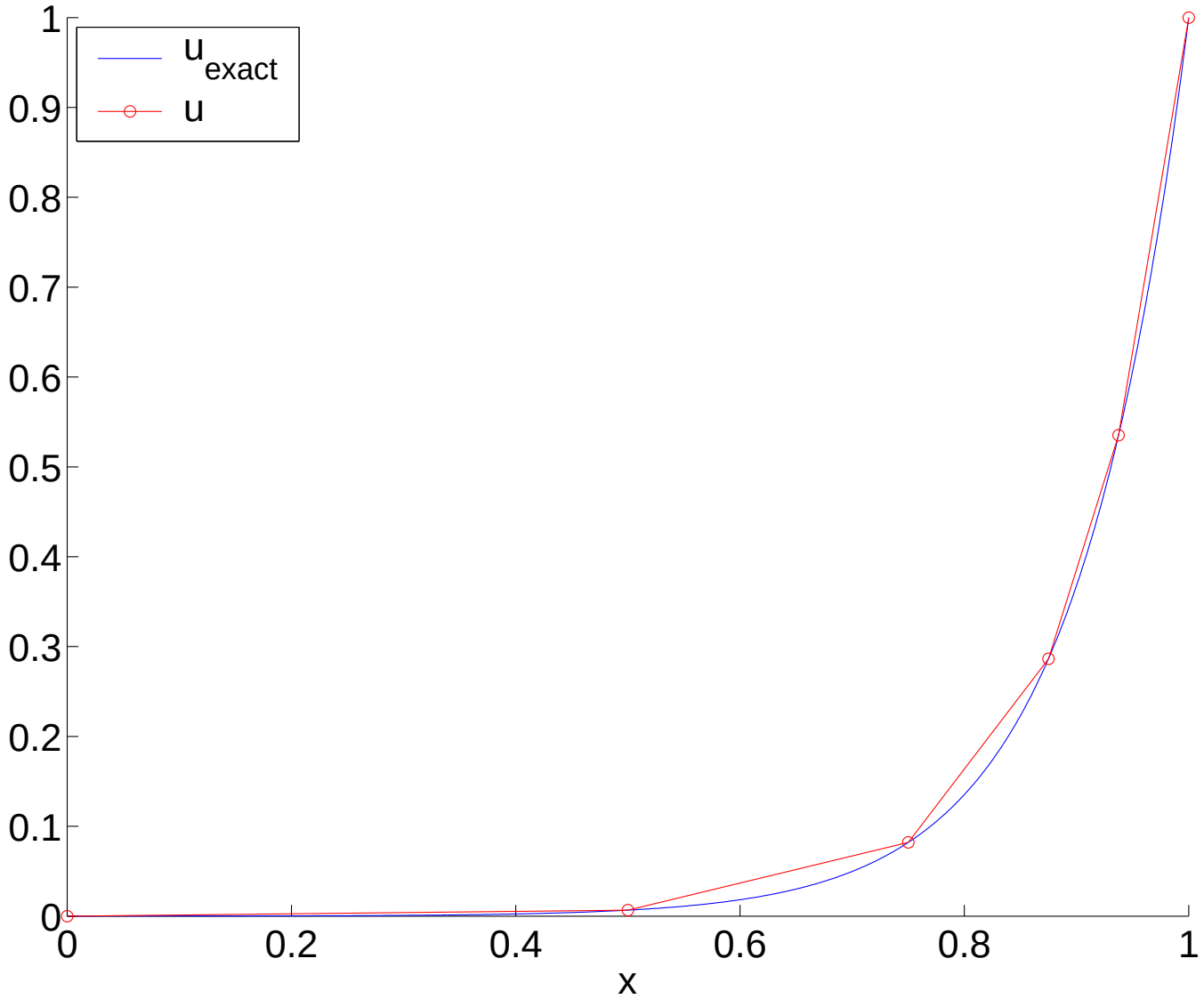
- A simple adaptive refinement strategy with `r_max` refinement steps for this 1D example problem is:

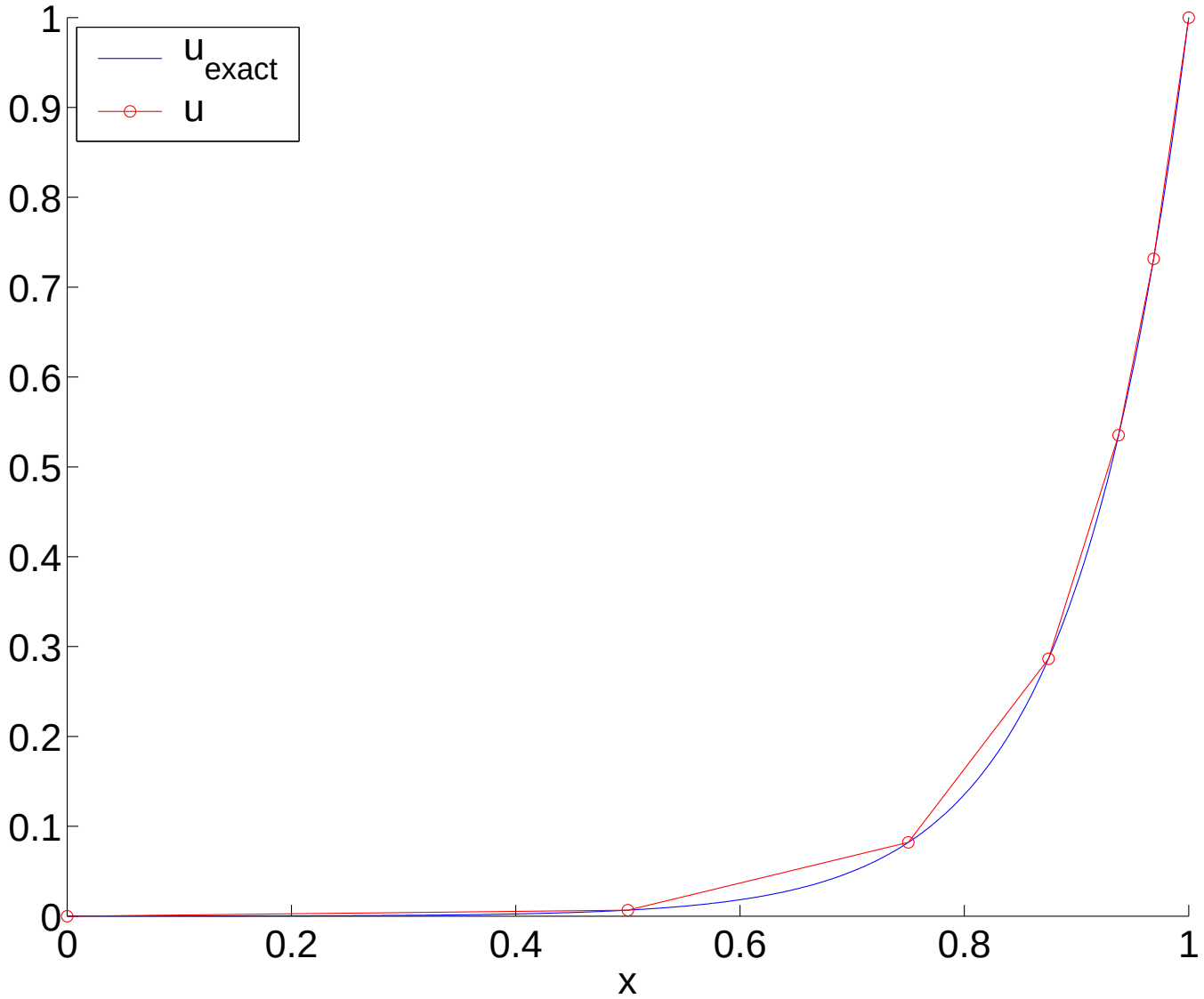
```
r=0;
while (r < r_max)
  Compute the FE solution (linear interpolant)
  Estimate the error (using flux-jump indicator)
  Refine the elements with error in top 10%
  Increment r
end
```

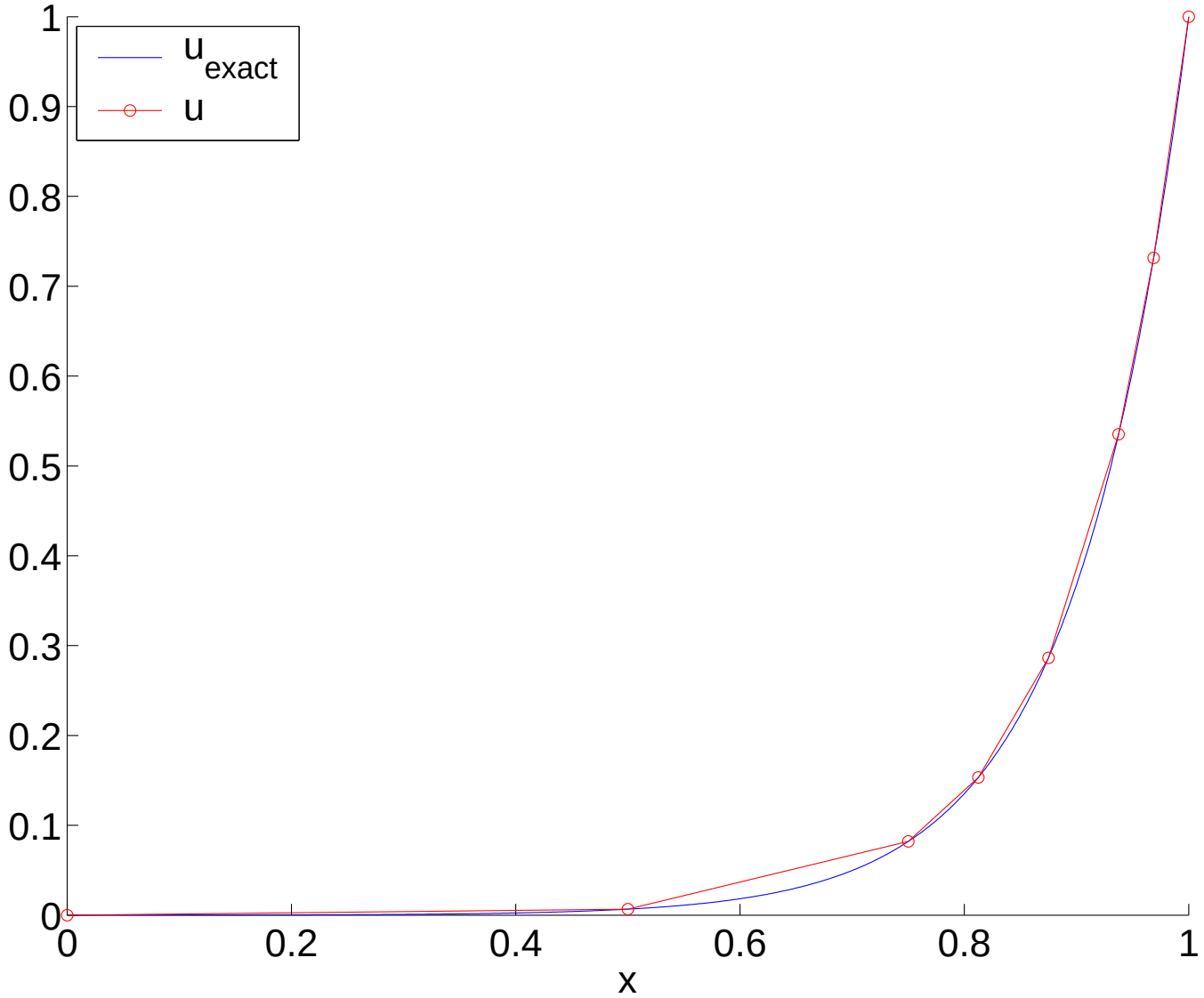

u exact
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
x

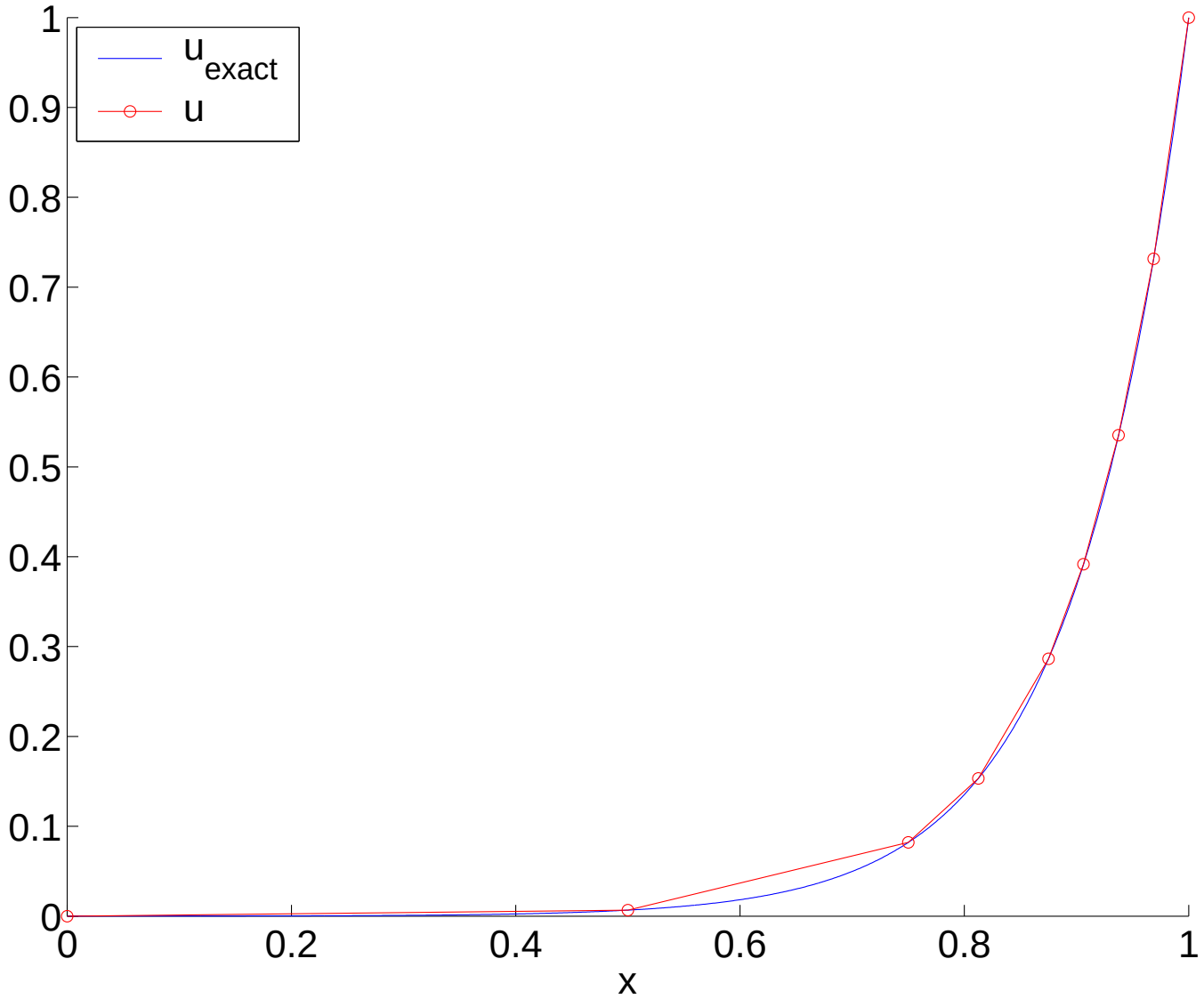

u exact
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
x

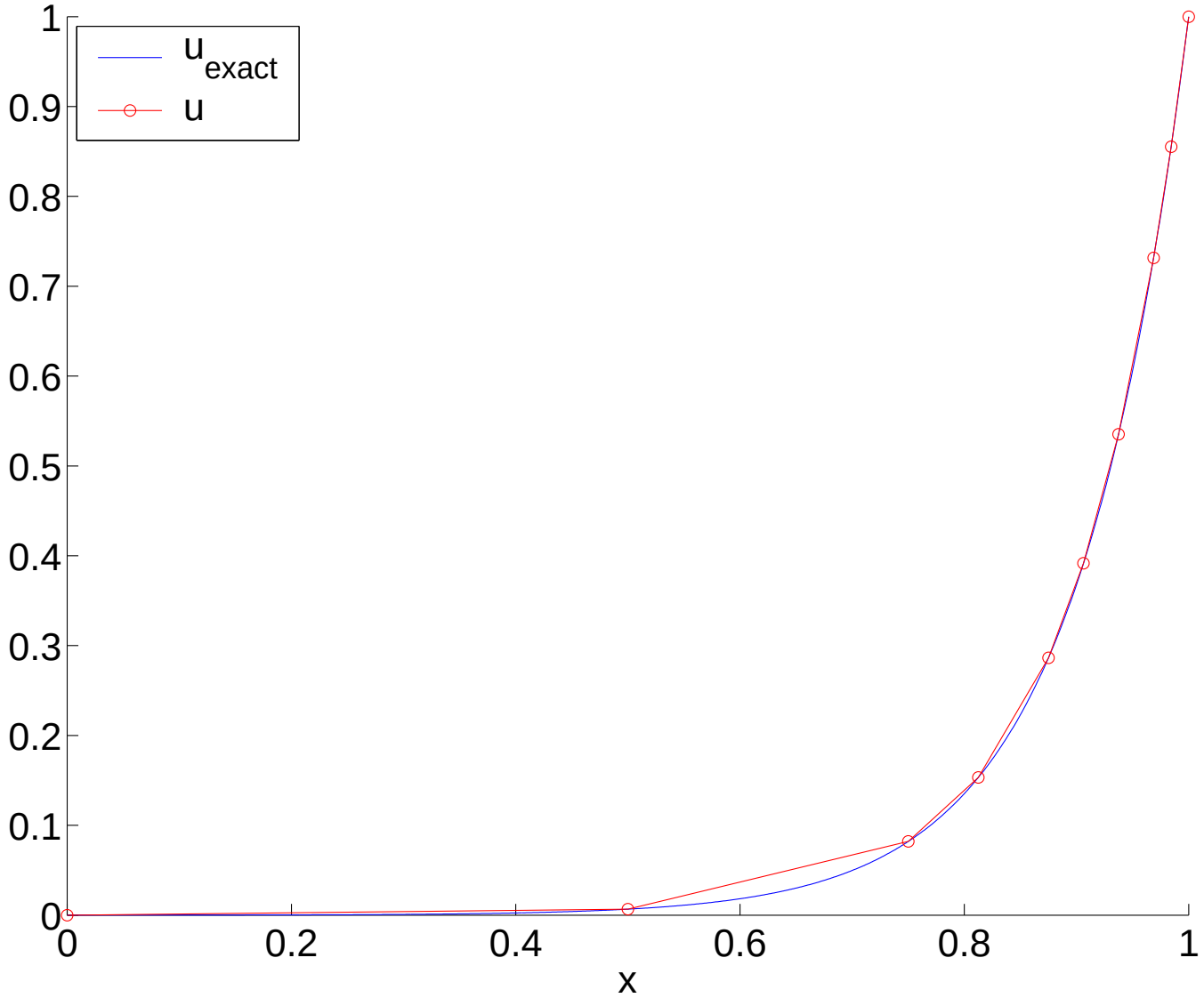

u exact
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
x

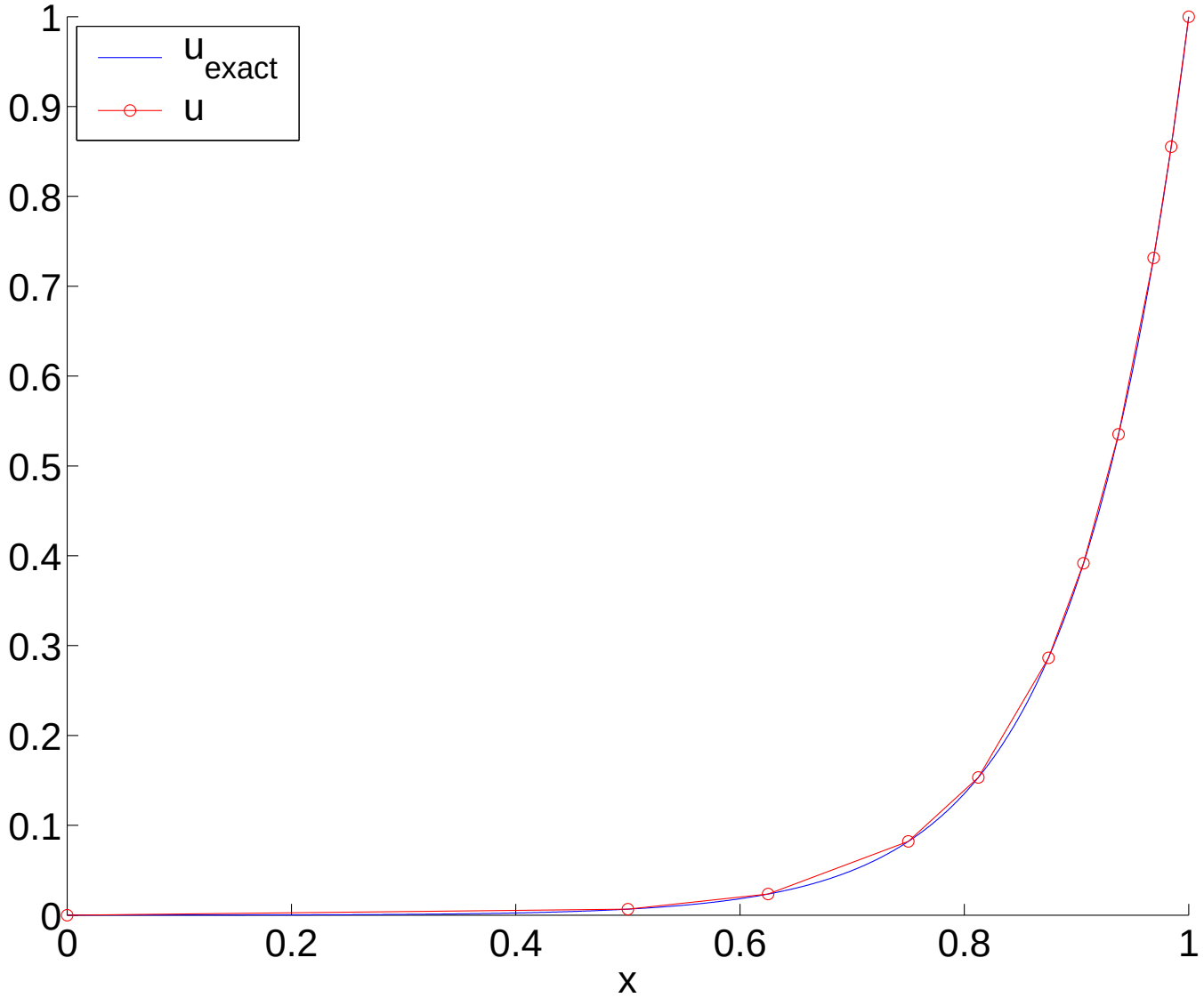

u exact
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
x

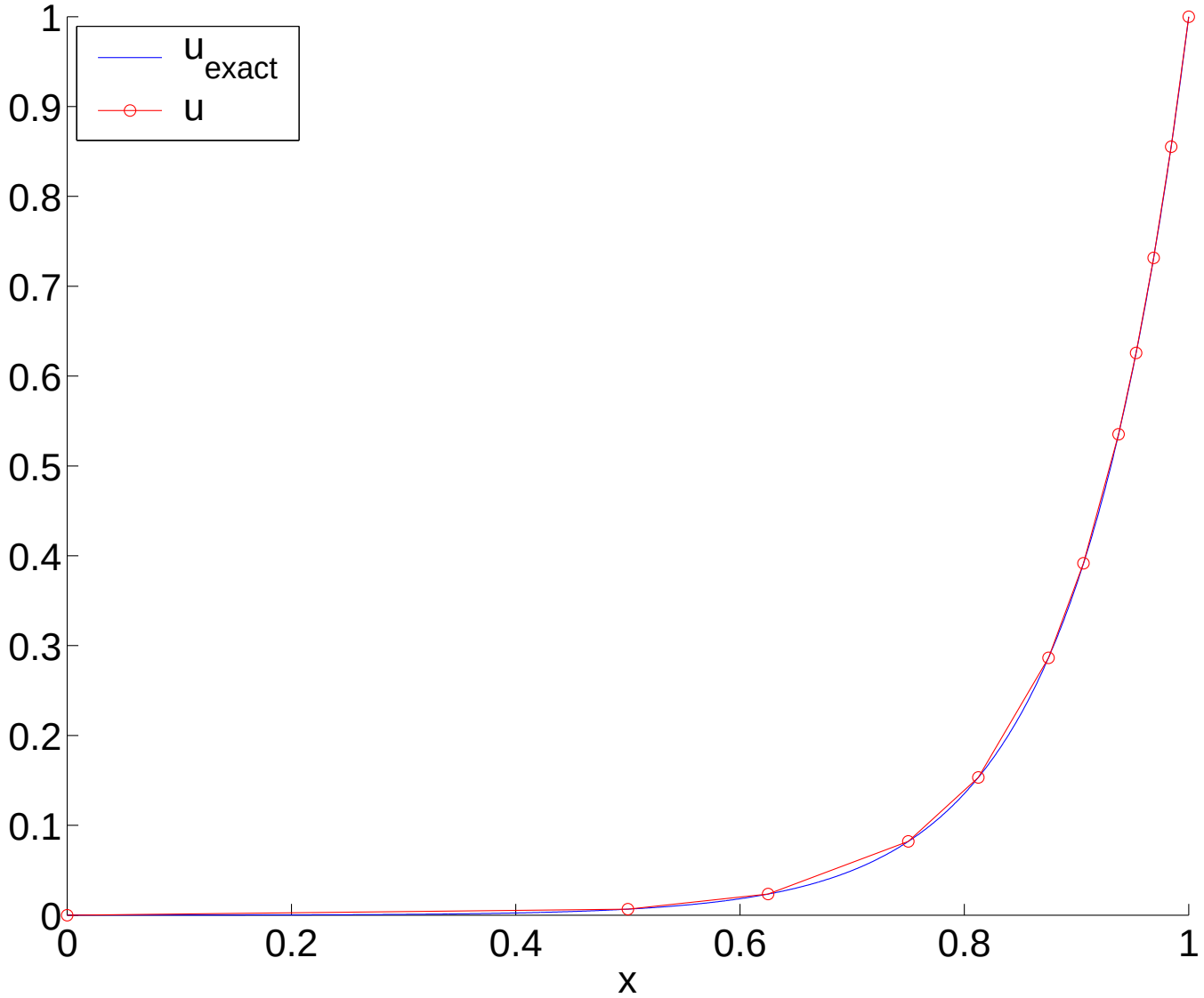

u exact
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
x

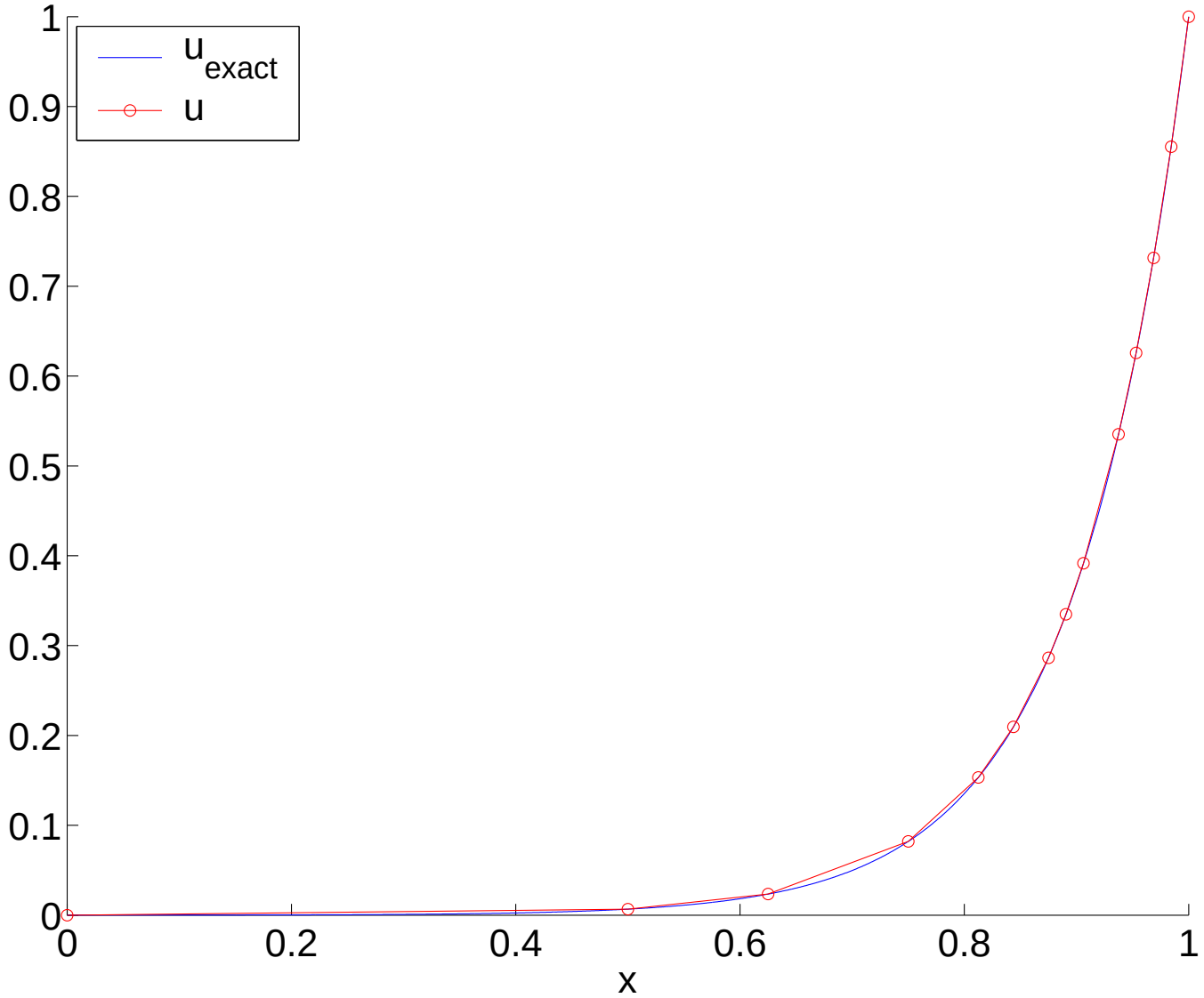

u exact
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
x


u exact
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
x

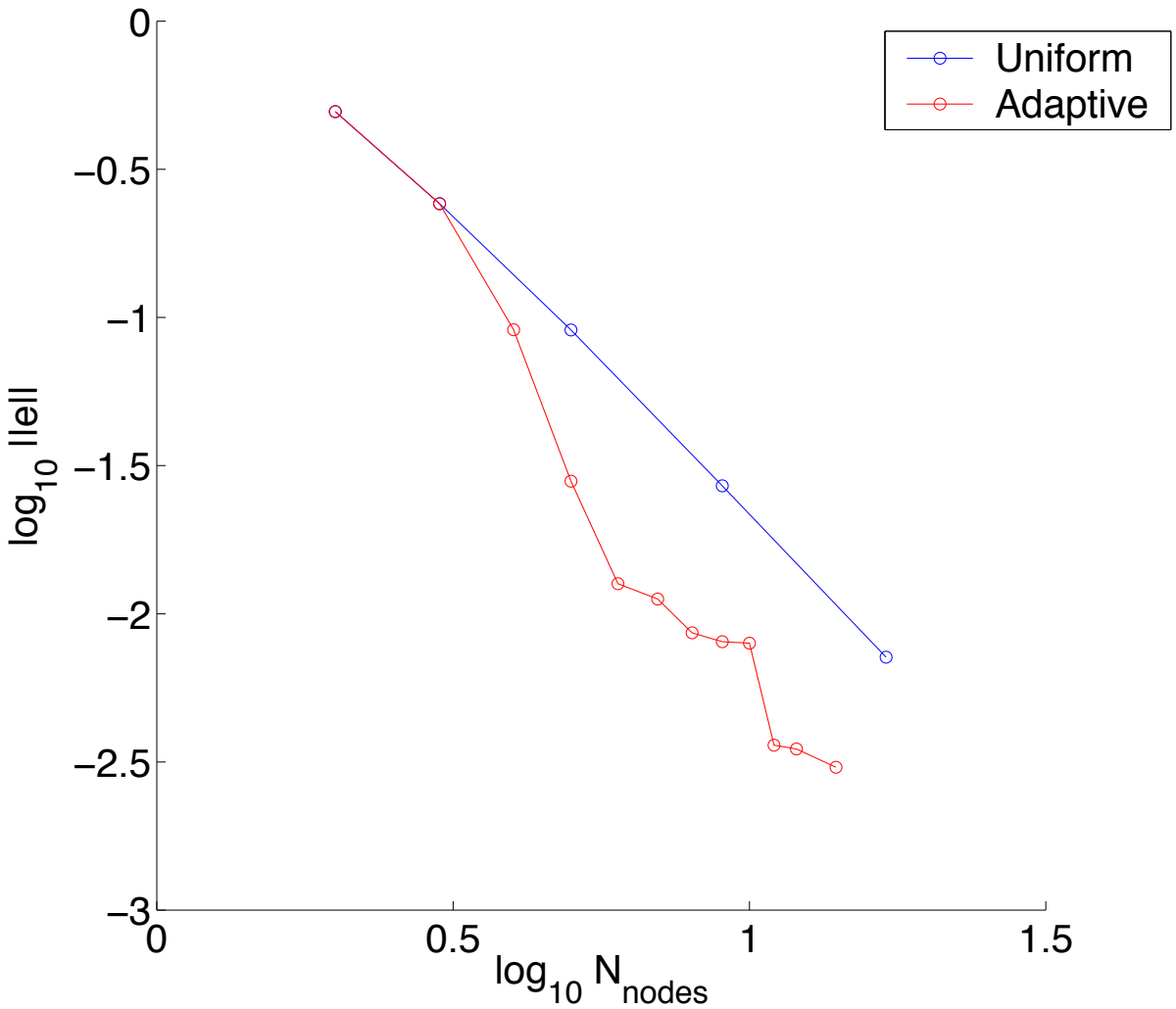

u exact
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
x


u exact
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
x


u exact
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
x

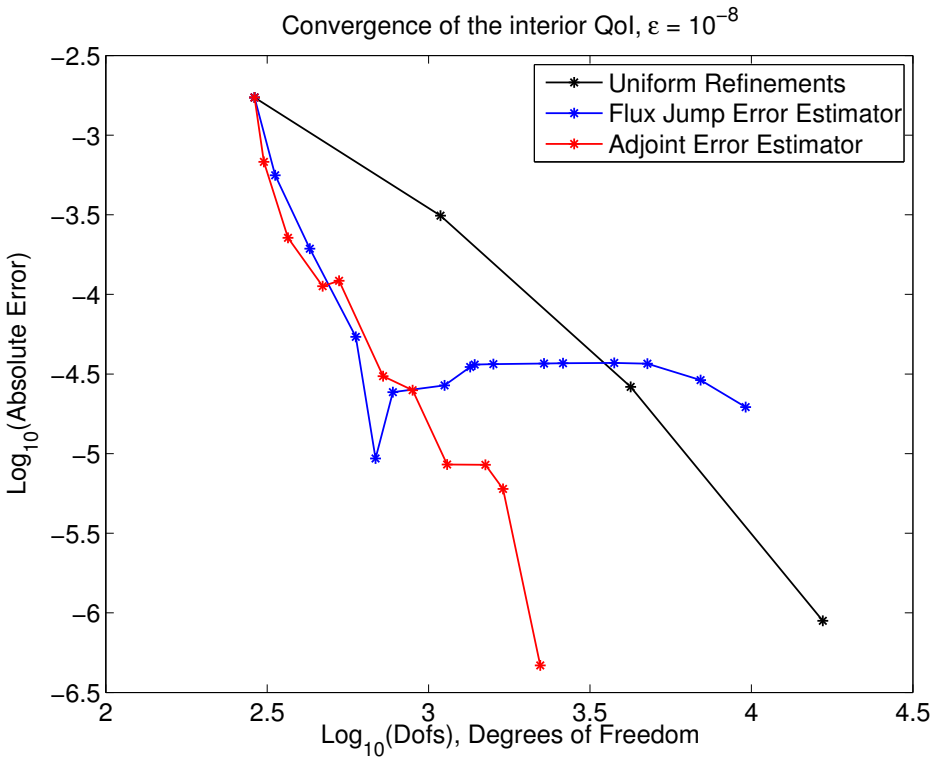

u exact
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
x

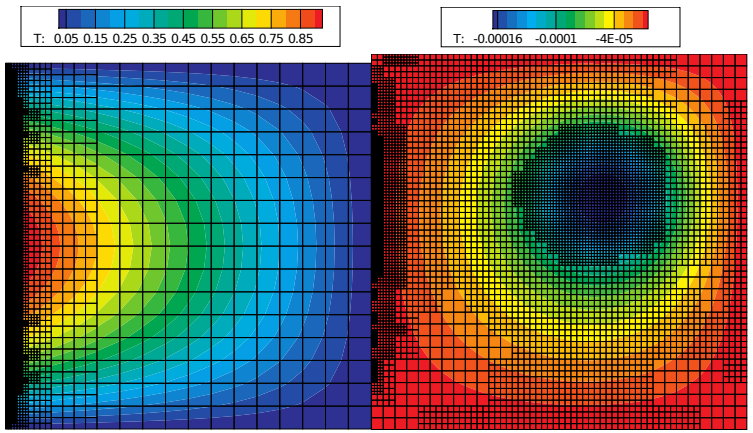
A Simple Refinement Strategy (cont.)



Goal-oriented Adaptivity

Refine to reduce solution error *when it influences QoI error*:





- Global error indicator targets layer alone; QoI temporarily plateaus
- Rapid convergence from adjoint-based refinement

Adaptivity: Error Estimators, Error Indicators

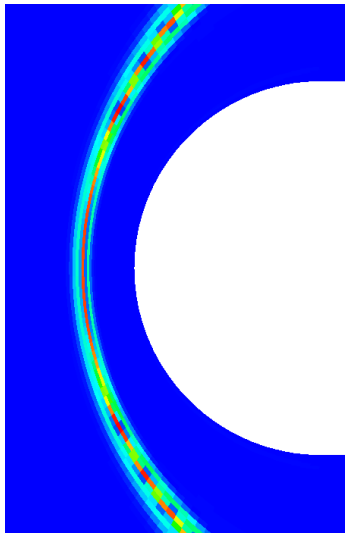
Error decompositions

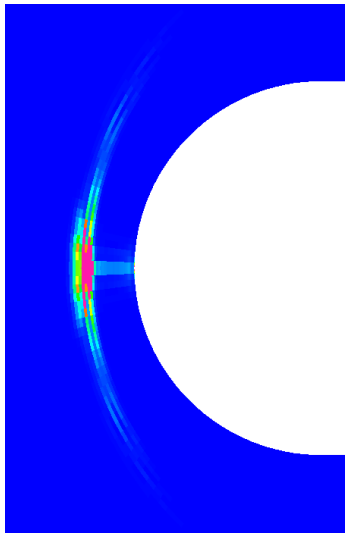
Subterms on each element K:

- $||u - u_h||^2_{\mathcal{H}} = \sum_K ||u - u_h||^2_{\mathcal{H}(\mathcal{K})} \leq \sum_K |\eta_K|^2$
- $Q(u) - Q(u_h) \approx \sum_K \eta_K$
- $|Q(u) - Q(u_h)| \leq \sum_K |\eta_K|$

Refinement heuristics

- Refinement/coarsening of elements with:
  - Worst/best fraction sorted by error
  - Error over/under fraction of tolerance
  - Error over/under target mesh size average
- h-vs-p refinement:
  - *a priori* singularity identification
  - Behavior vs. cost when h vs. p coarsening?





Adjoint-based Error Indicators, Sensitivities

$$\begin{aligned} Q(\tilde{u}^h) - Q(\tilde{u}) &= \mathcal{R}(\tilde{u}^h, \tilde{z} - \tilde{z}^h) + H.O.T. \\ q' &= Q_\xi(\tilde{u};\xi) - \mathcal{R}_\xi(\tilde{u}, \tilde{z} - \Psi(\tilde{u};\xi);\xi) \end{aligned}$$

Adjoint Refinement

$$\mathcal{R}(\tilde{u}^h, \tilde{z} - \tilde{z}^h) \approx \sum_E \mathcal{R}^E(\tilde{u}^h, \tilde{z}^H - \tilde{z}^h)$$

Adjoint Residual

$$\left|\mathcal{R}(\tilde{u}^h, \tilde{z} - \tilde{z}^h)\right| \leq \sum_E \left|\left|\mathcal{R}_u^E\right|\right|_{B(\mathbf{U}^E, \mathbf{V}^{E*})} \left|\left|\tilde{u} - \tilde{u}^h\right|\right|_{\mathbf{U}^E} \left|\left|\tilde{z} - \tilde{z}^h\right|\right|_{\mathbf{V}^E}$$

Generalized Adjoint Residual

$$\left|\mathcal{R}(\tilde{u}^h, \tilde{z} - \tilde{z}^h)\right| \leq \sum_E \left|\left|\tilde{z}_i - \tilde{z}^h{}_i\right|\right|_i \; M_{ij} \; \left|\left|\tilde{u}_j - \tilde{u}^h{}_j\right|\right|_j$$

Outline

Parallelism Goals

Reduced CPU time

- Distributed CPU usage
- Asynchronous I/O

Reduced memory requirements

- Larger attainable problem size

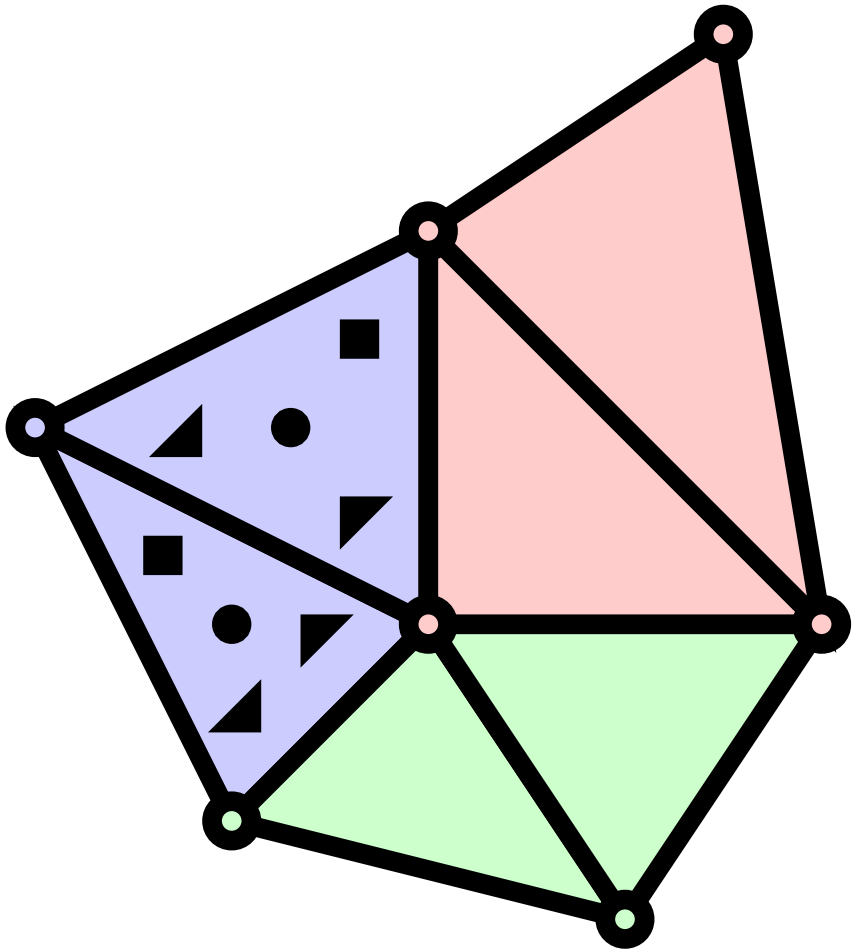
Parallelism Levels

SIMD Instructions

- Assemblies, MatVec operations

Shared-memory Threads

- Mesh Partitioning



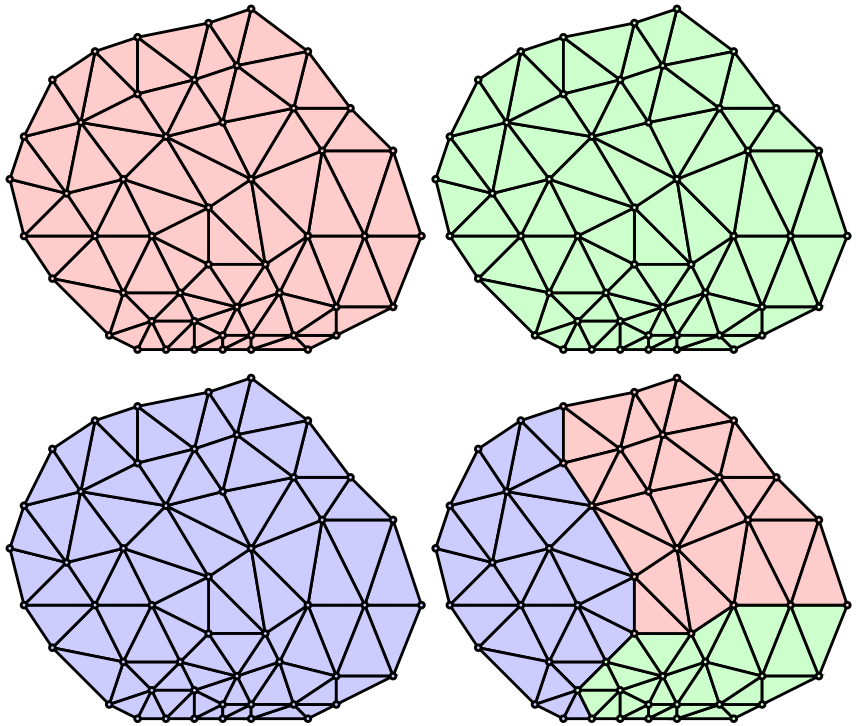
Parallelism Levels

Separate Simulations

- Parametric studies
- Uncertainty analysis

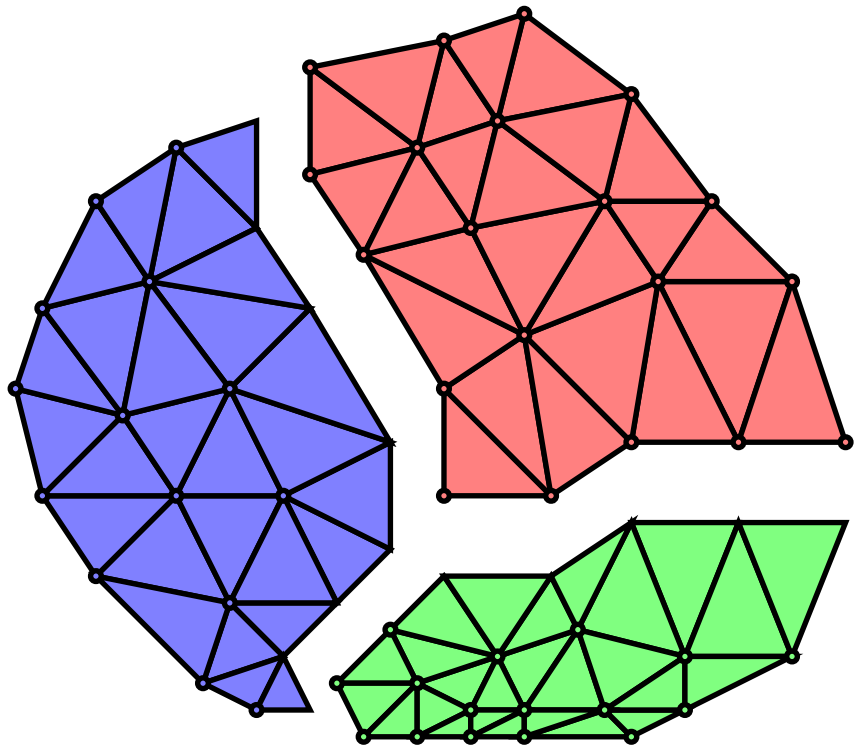
Distributed-memory Processes

- Asynchronous I/O
- Mesh Partitioning



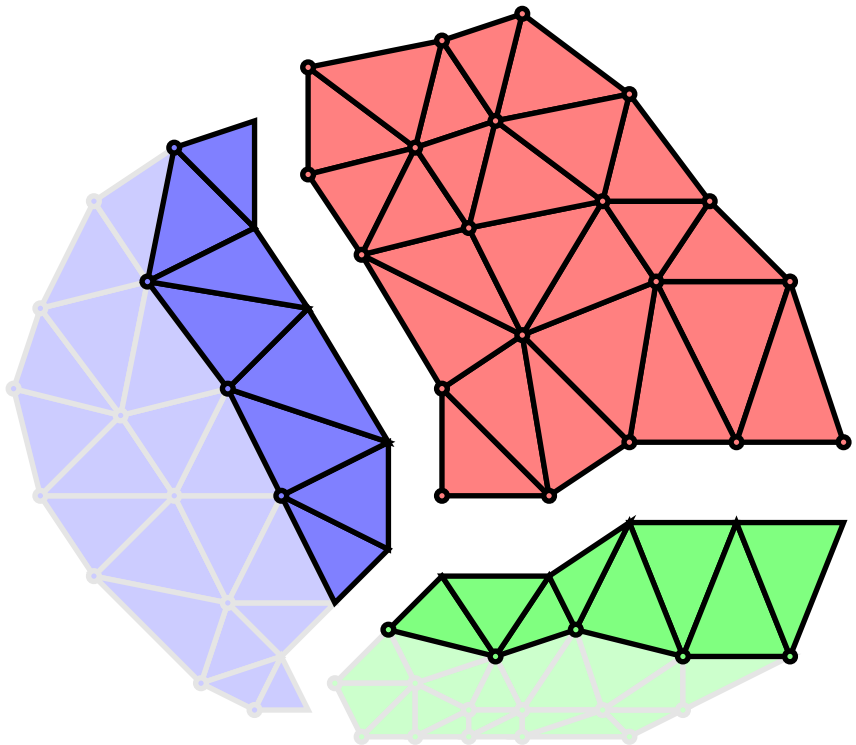
SerialMesh Partitioning

- Each element, node is “local” to one processor
- Each processor has an identical Mesh copy
- Mesh stays in sync through redundant work
- FEM data synced on “ghost” elements only



ParallelMesh Partitioning

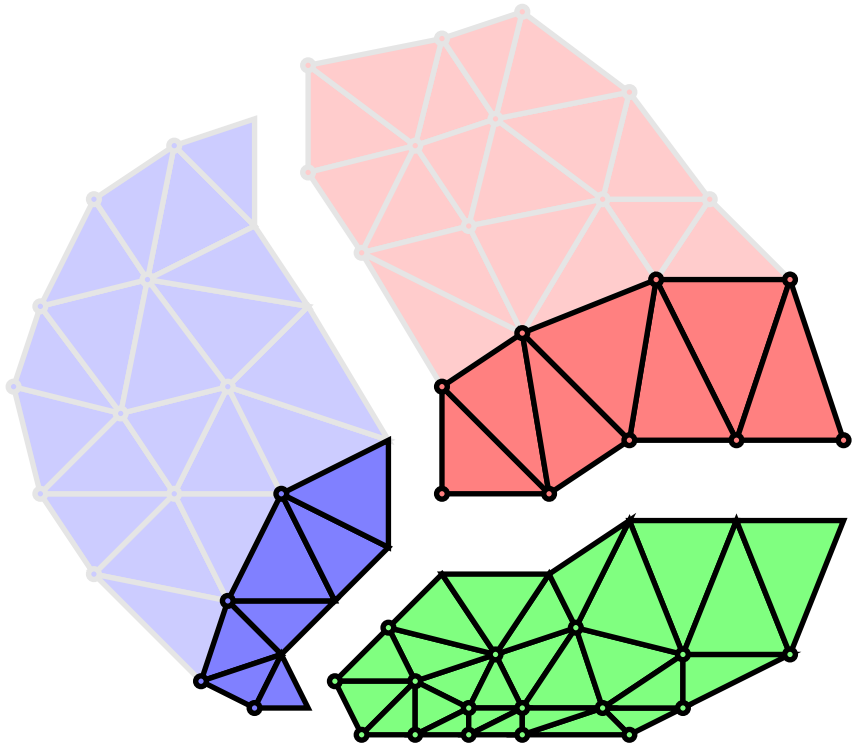
- Processors store only local and ghost objects
- Each processor has a small Mesh subset
- Mesh stays in sync through MPI communication



ParallelMesh Partitioning

Pros

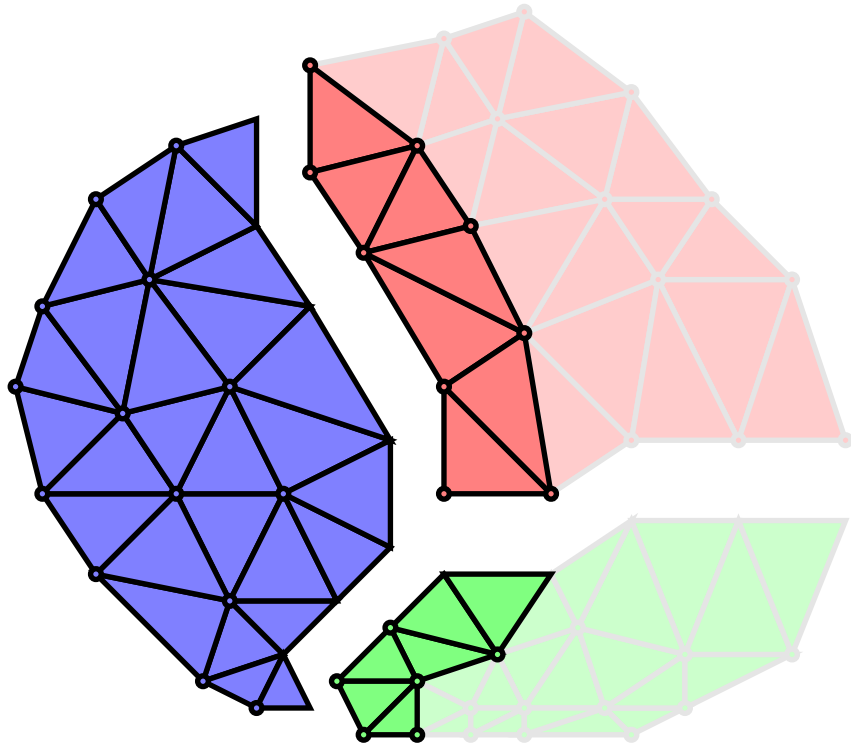
- Reduced memory use
- $O(N_E/N_P)$ CPU costs



ParallelMesh Partitioning

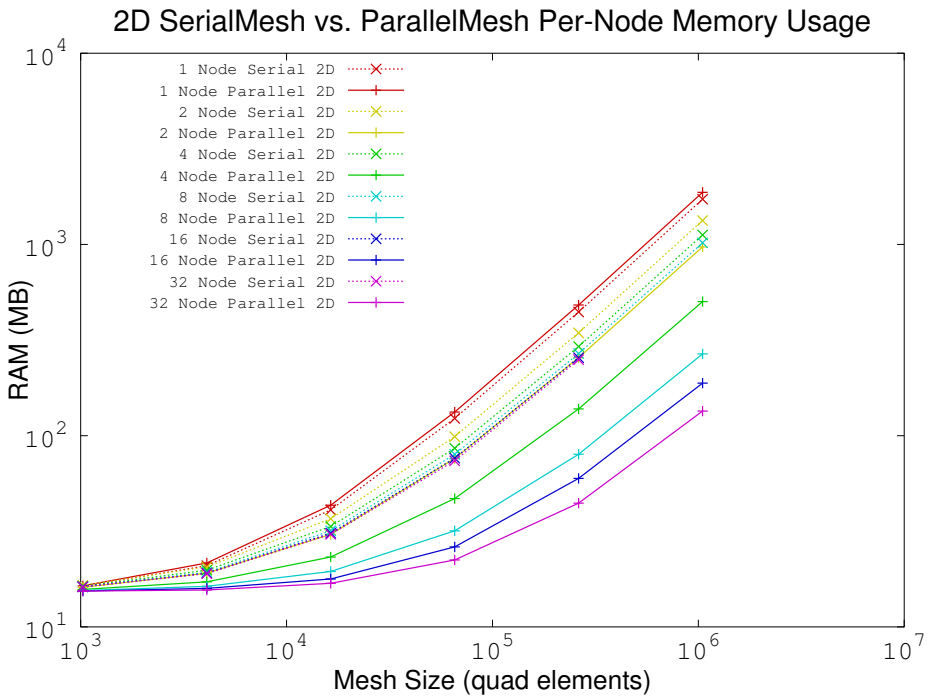
Cons

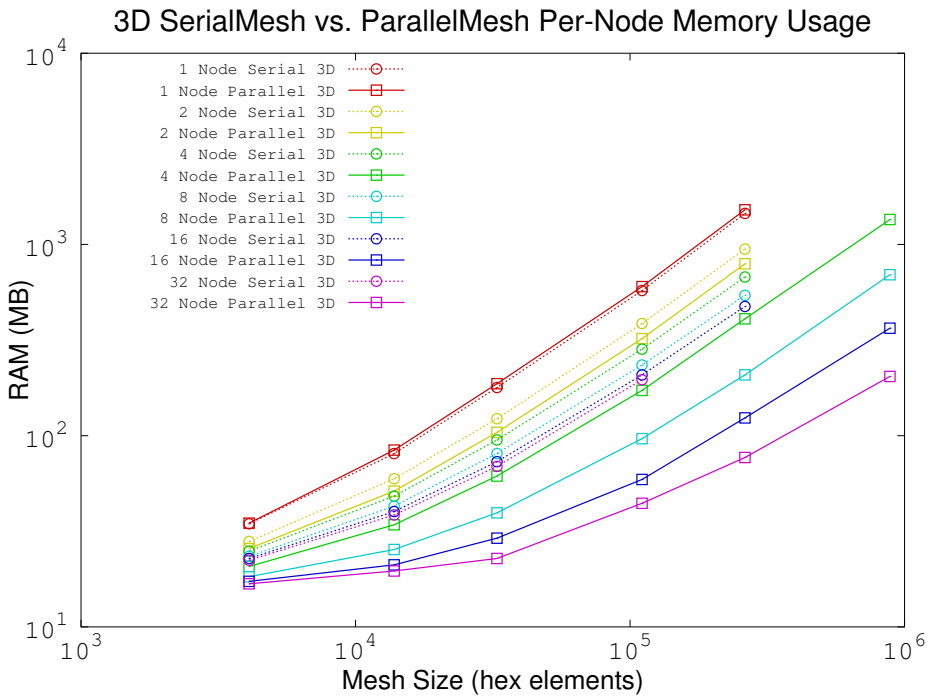
- Increased code complexity
- Increased synchronization "bookkeeping"
- Greater debugging difficulty



“Typical” PDE example

Transient Cahn-Hilliard, Bogner-Fox-Schmidt quads or hexes



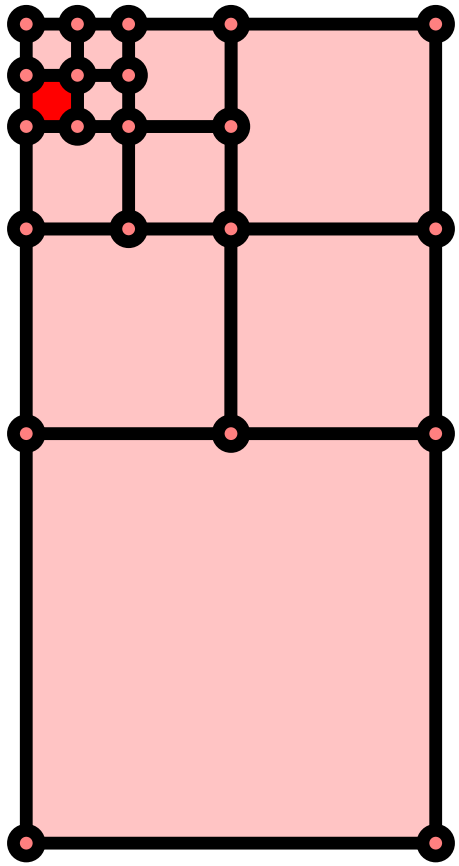


Results

- Parallel codes using SerialMesh are unchanged for ParallelMesh
- Overhead, distributed sparse matrix costs are unchanged
- Serialized mesh, indexing once dominated RAM use

Distributed Mesh Refinement

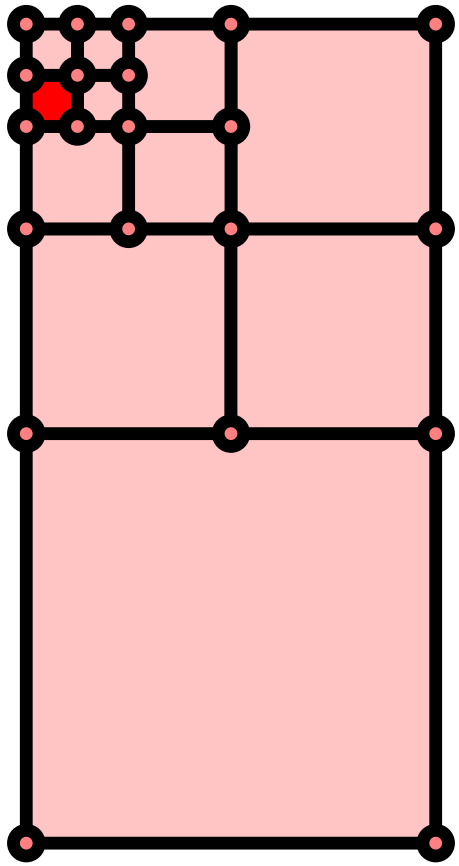
Elem, Node creation



Distributed Mesh Refinement

Elem, Node creation

- Ids $\{i : i \bmod (N_P + 1) = p\}$ are owned by processor p



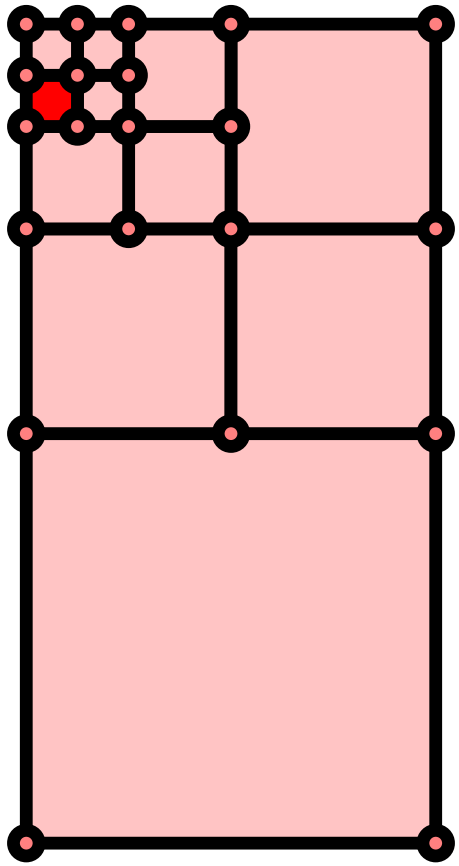
Distributed Mesh Refinement

Elem, Node creation

- Ids $\{i : i \bmod (N_P + 1) = p\}$ are owned by processor p

Synchronization

- Refinement Flags
- New ghost child elements, nodes
- Hanging node constraint equations



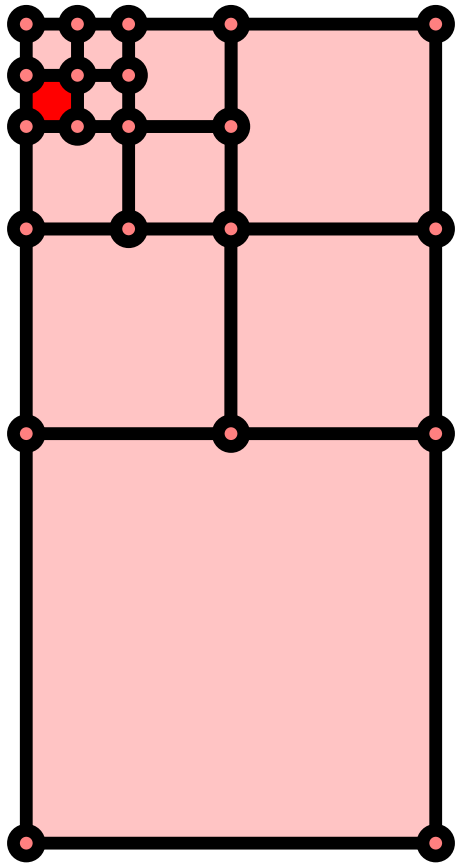
Distributed Mesh Refinement

Elem, Node creation

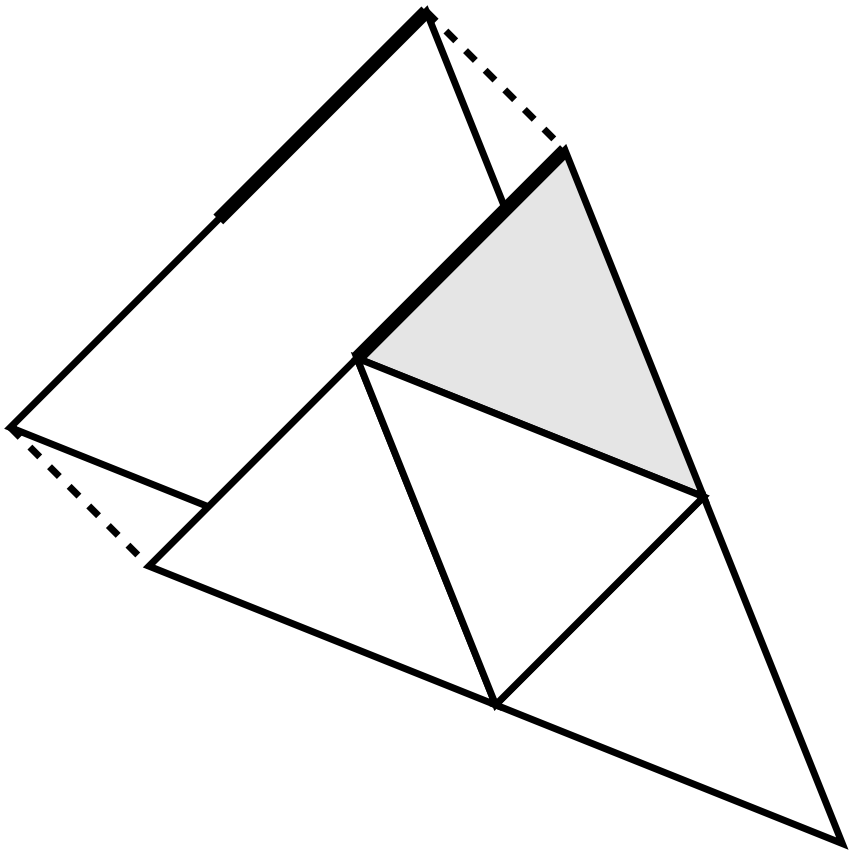
- Ids $\{i : i \bmod (N_P + 1) = p\}$ are owned by processor p

Synchronization

- Refinement Flags
  - Data requested by id
  - Iteratively to enforce smoothing
- New ghost child elements, nodes
  - Id requested by data
- Hanging node constraint equations
  - Iteratively through subconstraints, subconstraints-of-subconstraints...



Adaptivity and ParallelMesh



Challenges

- Reindexing elements, nodes, DoFs
- Synchronization of ghost objects
- Load balancing, Repartitioning

ParallelMesh Data Structure

std::vector fails

- Not sparse
- $O(N_E)$ storage cost

std::map

- "mapvector" interface provides iterators
- $O(log(N_E/N_P))$ lookup time without std::unsorted_map
- $O(1)$ lookup time with std::unsorted_map

Parallel:: API

Encapsulating MPI

- Improvement over MPI C++ interface
- Makes code shorter, more legible

Example:

```
std::vector<Real> send, recv;
...
send_receive(dest_processor_id, send,
             source_processor_id, recv);
```

Parallel:: API

Instead of:

```
if (dest_processor_id   == libMesh::processor_id() &&
    source_processor_id == libMesh::processor_id())
  recv = send;
#ifdef HAVE_MPI
else
  {
    unsigned int sendsize = send.size(), recvsize;
    MPI_Status status;
    MPI_Sendrecv(&sendsize, 1, datatype<unsigned int>(),
                 dest_processor_id, 0,
                 &recvsize, 1, datatype<unsigned int>(),
                 source_processor_id, 0,
                 libMesh::COMM_WORLD,
                 &status);

    recv.resize(recvsize);

    MPI_Sendrecv(sendsize ? &send[0] : NULL, sendsize, MPI_DOUBLE,
                 dest_processor_id, 0,
                 recvsize ? &recv[0] : NULL, recvsize, MPI_DOUBLE,
                 source_processor_id, 0,
                 libMesh::COMM_WORLD,
                 &status);
  }
#endif // HAVE_MPI
```

Outline

Verification: Finding The Unknown Unknowns

Solution Verification

Estimating Numerical Error:

- Discretization Error
- Iterative Error
- Round-off Error

These errors cannot be avoided, but can be quantified.

Code Verification

- Software bugs
- Numerical algorithm weaknesses
- Model implementation mistakes

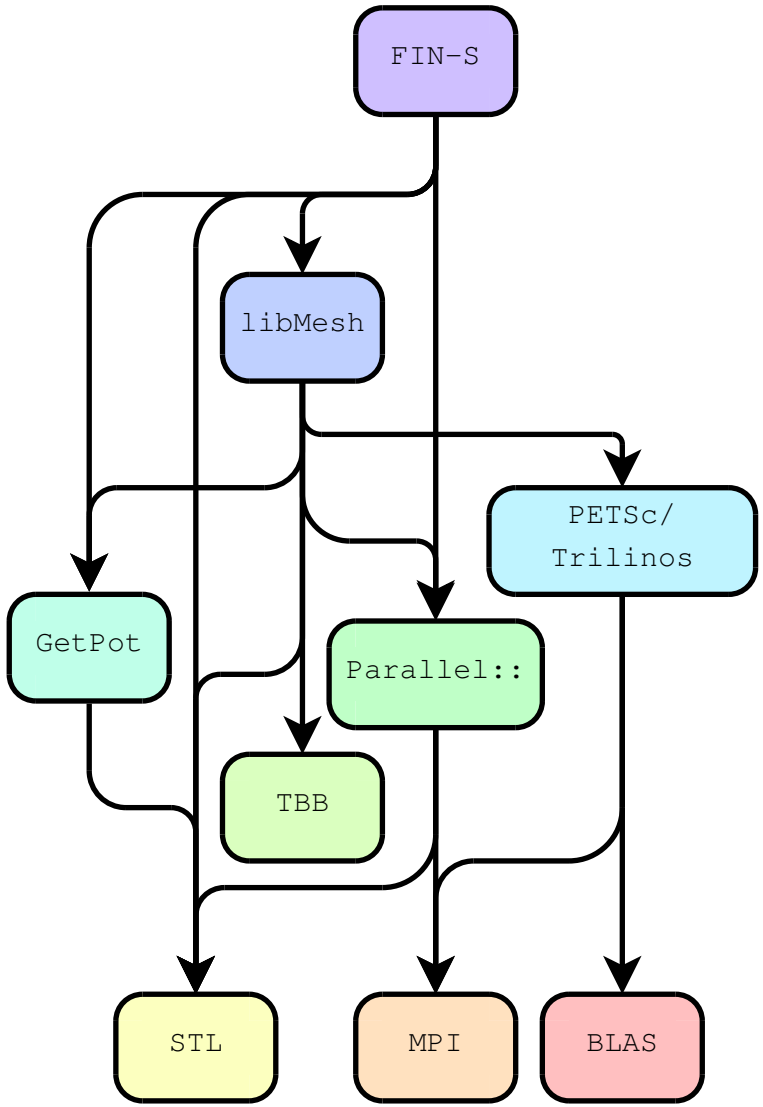
Codes cannot practically be proven error-free, but can be proven to have errors. So we try our best to do the latter...

Code Reuse

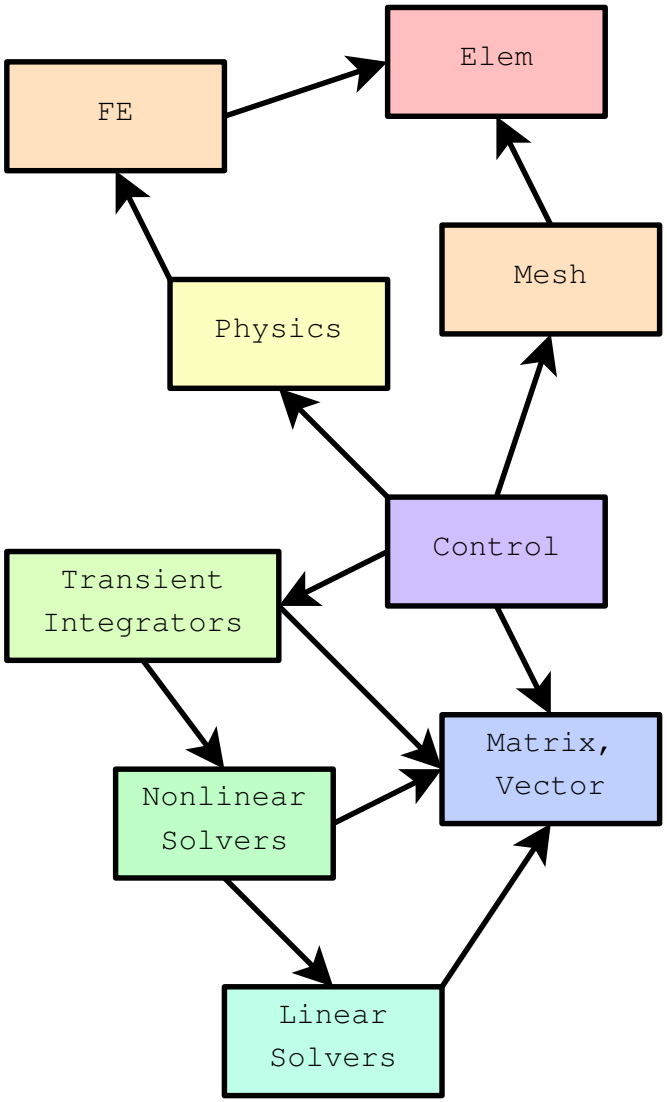
Examples

- MPI, BLAS
- Aztec, MUMPS, PETSc
- deal.II, FEniCS, libMesh

- Don't reinvent the wheel unnecessarily!
- Time spent rewriting something old is time that could have been spent writing something new.
- More eyes == fewer bugs
- Extend existing capabilities where possible.



Modular Programming



Discrete Components, Interfaces

- Linear, nonlinear solvers are discretization-independent
- System assembly, solution I/O & postprocessing can be discretization-independent
- Time, space discretizations should be physics-independent
- Error analysis, sensitivity methods can be physics-independent

- Reusable components get re-tested
- Errors too subtle to find in complex physics are easy to spot in benchmark problems.

Assertions

“When in trouble when in doubt, run in circles scream and shout.”
- old Army War College football team slogan

Assertions

“When in trouble when in doubt, run in circles scream and shout.”
- old Army War College football team slogan

```
#if IN_DOUBT {
  if (in_trouble()) {
    run_in_circles();
    scream_and_shout();
  }
#endif
```

Assertions

“When in trouble when in doubt, run in circles scream and shout.”
- old Army War College football team slogan

```
#if IN_DOUBT {
  if (in_trouble()) {
    run_in_circles();
    scream_and_shout();
  }
#endif
```

```
assert(!in_trouble());
```

High-level Assertions

`libmesh_assert()`, PETSc debug mode

- Active only in “debug” runs
- Function preconditions
  - Are function arguments all valid?
- Function postconditions
  - Does function result satisfy requirements?
- Approx. 7000 assertions in `libMesh` alone

Assertion types

Standard `assert()` prints failed assertion, prints file/line numbers, exits
`libmesh_assert()` adds:

- Per-processor stack trace files
  - Line numbers via gdb
- C++ exception thrown

High-level Assertions Examples

```
libmesh_assert(c < _variables.size());
libmesh_assert(s < elem->n_sides());
libmesh_assert((ig >= Ug.first_local_index()) &&
               (ig < Ug.last_local_index()));
libmesh_assert(requested_ids[p].size() == ghost_objects_from_proc
libmesh_assert(obj_procid != DofObject::invalid_processor_id);
MeshTools::libmesh_assert_valid_node_procids(mesh);
libmesh_assert(neigh->has_children());
libmesh_assert(this->closed());
libmesh_assert(this->initialized());
libmesh_assert(mesh.is_prepared());
libmesh_assert(error_estimator.error_norm.type(var) == H1_SEMINORM
               error_estimator.error_norm.type(var) == W1_INF_SEMI
libmesh_assert(number_h_refinements > 0 || number_p_refinements >
```

Low-level Assertions

Defining `_GLIBCXX_DEBUG`

- Runtime bounds-checking of standard `vector`, `iterator` use
- Similar to ifort "-check bounds"
- Out Of Bounds errors otherwise lead to corrupt data, not just segfaults!

Unit Tests

Testing One Object At A Time

- Reusable modules interact with all other code through a limited API
- That API can be tested directly outside of application code
- Test one method at a time, isolate problems locally
- 108 unit tests currently in `libMesh`

Unit Tests

Testing One Object At A Time

- Reusable modules interact with all other code through a limited API
- That API can be tested directly outside of application code
- Test one method at a time, isolate problems locally
- 108 unit tests currently in `libMesh`

Tests are useful **only when run**!

- Regression, example, application tests had more coverage
- Unit tests became useful only when added to CI
- Most useful: Test-driven development
  - Write tests first
  - Write code to fit

Unit Tests Example

```
#include <quadrature.h>

class QuadratureTest : public CppUnit::TestCase {
public:
  CPPUNIT_TEST_SUITE( QuadratureTest );
  CPPUNIT_TEST( test3DWeights<FIFTH> );  // etc.

  template <Order order>
  void test3DWeights ()
  {
    AutoPtr<QBase> qrule = QBase::build(QGAUSS, 3, order);
    qrule->init (TET10);
    sum = 0;
    for (unsigned int qp=0; qp<qrule->n_points(); qp++)
      sum += qrule->w(qp);
    CPPUNIT_ASSERT_DOUBLES_EQUAL( 1./6., sum , TOLERANCE*TOLERANCE
  }
};
```

Parametric Testing

One Test Code, Many Tests

- Keep test codes generic
- Execute with many different parameter choices
- `libMesh` compile time examples:
  - Algebraic solver interface
  - Real/complex arithmetic
  - Mesh data structure
- `libMesh` run time examples:
  - Geometric element type
  - Finite element type
  - Polynomial degree
  - Error indicator type
  - Processor count
  - Partitioner
  - Adaptive refinement strategy
  - I/O format

Regression Tests

Revise Software ⇒ Rerun Tests

- Example applications, unit tests, benchmark tests
- Catches *unintended consequences* of revisions
- Continuous Build System automation
  - Tests once run “by hand” by `libMesh` developers
  - BuildBot tests for wide configuration coverage
  - Civet tests on every Pull Request

Verification Benchmark Problems

Choosing Test Problems

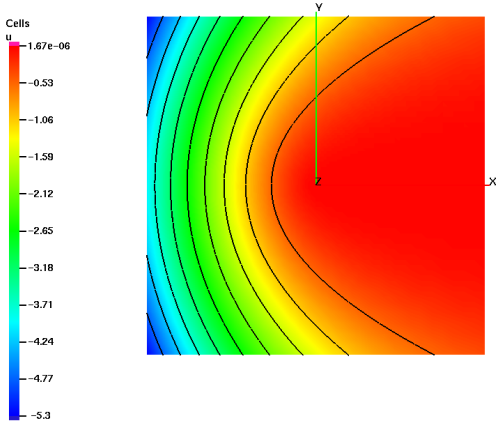
Capitalize on anything you know a priori:

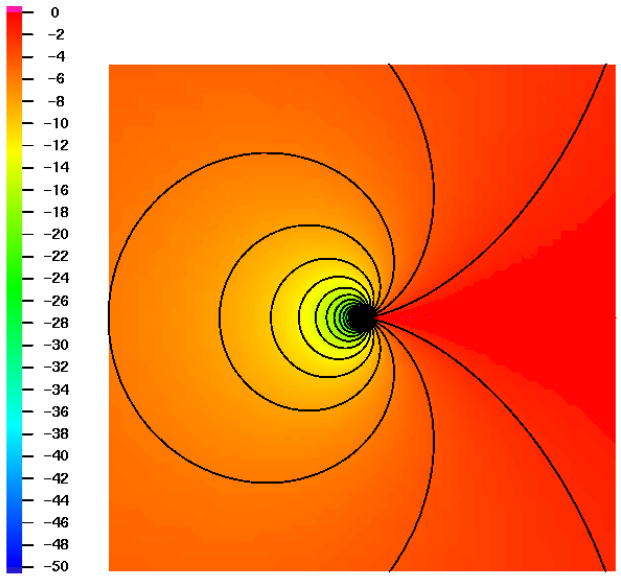
- Known solutions
  - Exact solution to discretized problem
  - Limit solution of continuous problem
  - Known quantities of interest
- Known asymptotic convergence rates
- **Known residuals**

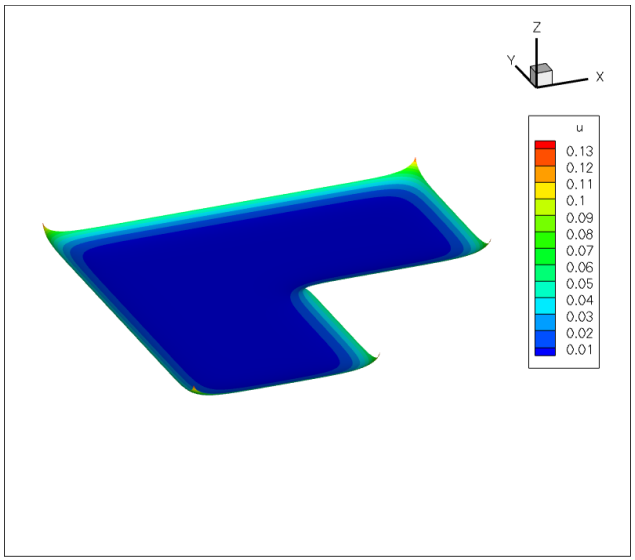
Known Solutions, Functionals

Examples

- Incompressible flow around a cusp
- Wetting angle in Laplace-Young surface tension







Manufactured Solutions

Any Solution for Any Equation

- Real Physics: $\boldsymbol{R}(\boldsymbol{u}) = 0$ for $\boldsymbol{u} = \boldsymbol{u}^r$
- Choose manufactured solution $\boldsymbol{u}^m$
- Desired Physics: $\boldsymbol{R}^m(\boldsymbol{u}) = 0$ for $\boldsymbol{u} = \boldsymbol{u}^m$
- Construct $\boldsymbol{R}^m(\boldsymbol{u}) \equiv \boldsymbol{R}(\boldsymbol{u}) - \boldsymbol{R}(\boldsymbol{u}^m)$

Manufactured Solution Example

Convection-Diffusion Problem with Adjoints

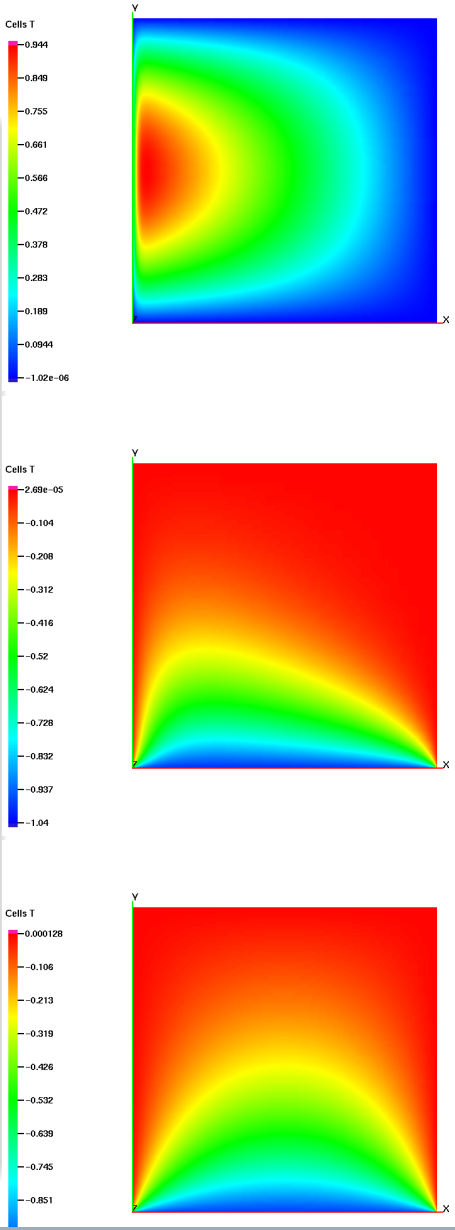
Residual equation:

$$\boldsymbol{R}(\boldsymbol{u}) = \nabla \cdot \alpha \nabla \boldsymbol{u} + \beta \vec{e}_x \cdot \nabla \boldsymbol{u} + \boldsymbol{f} = 0$$

Manufactured solution:

$$\boldsymbol{u} \equiv 4(1 - e^{-\alpha x} - (1 - e^{-\alpha})x)y(1 - y)$$

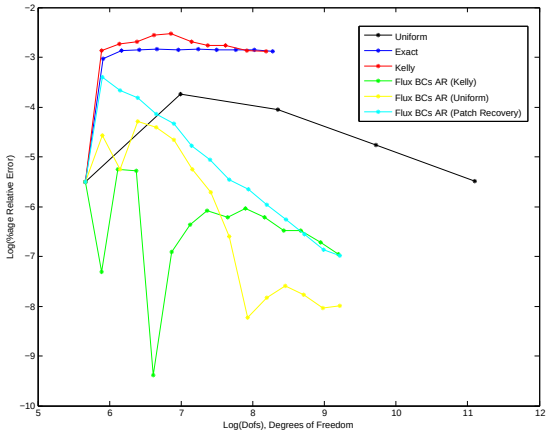
- Homogeneous Dirichlet boundary
- α controls flux strength, layer
- Choose any convection strength β, solve for $\boldsymbol{f}$
- $\beta = 0$ gives simple series adjoint solutions

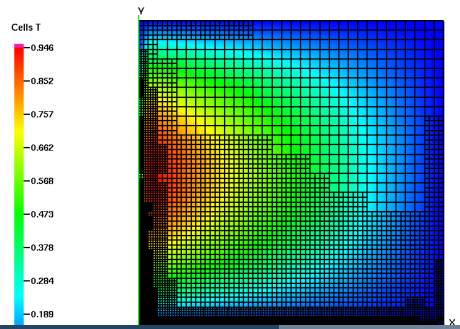


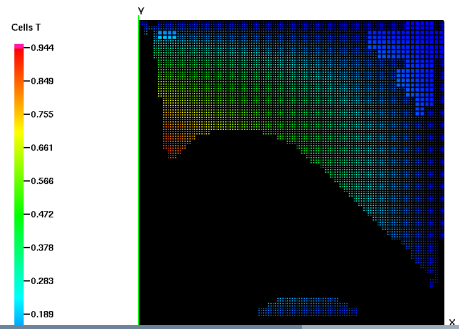
Manufactured Solution Example

Goal-Oriented Refinement

- Superconvergence on some grids
- Convergence "plateaus" found in multiple refinement strategies
- `UniformRefinementEstimator` required new code to solve for adjoint solution errors
- `PatchRecoveryErrorEstimator` required new seminorm integration (H^1/L_2/mixed vs. $W^{1,\inf}$) to give compatible error subestimates



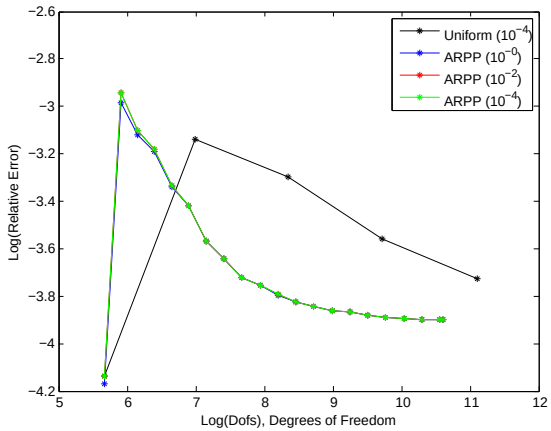




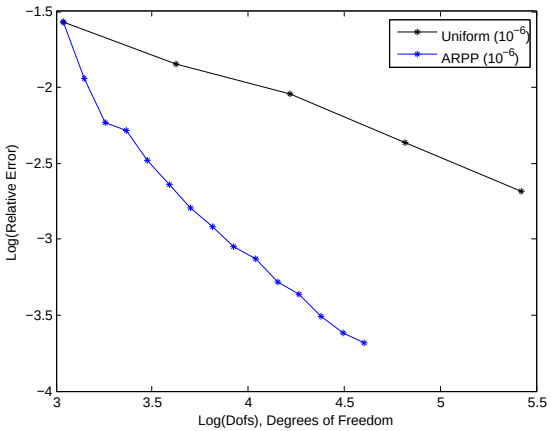
Manufactured Solution Example

Adjoint-based Parameter Sensitivity

- Convergence to analytic sensitivity plateaus at 2% relative error in **every** refinement strategy
- Finite differenced partial derivatives not responsible
- Manufactured solution allowed sensitivity subcomponent comparison to analytic solutions
- Sign errors in `libMesh` parameter sensitivity method



Manufactured Solution Example



Adjoint-based Parameter Sensitivity

- "Off by 100%" error remaining in one subterm of equations
- Switch to $u'' = f$, 1D quadratic solutions, manufactured residual test
- Identified bug in repeated `adjoint_solve` rhs assembly
- Returned to manufactured solution benchmark: now converges to true solution

Manufactured Solution Issues

Compressible Inviscid Perfect Gas Flow

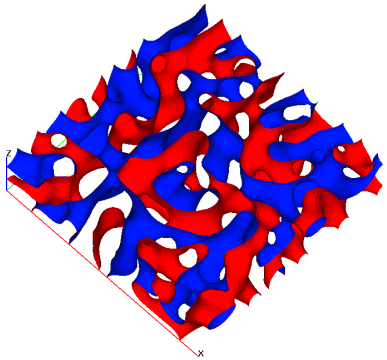
$$
\begin{aligned}
Q_{e_t} = &-\frac{\gamma}{\gamma-1}\frac{\pi a_{px} p_x}{L}\sin\left(\frac{\pi x a_{px}}{L}\right)\left[u_x\sin\left(\frac{\pi x a_{ux}}{L}\right)+u_y\cos\left(\frac{\pi y a_{uy}}{L}\right)+u_z\cos\left(\frac{\pi z a_{uz}}{L}\right)+u_0\right]+\\
&+\frac{\gamma}{\gamma-1}\frac{\pi a_{py} p_y}{L}\cos\left(\frac{\pi y a_{py}}{L}\right)\left[v_x\cos\left(\frac{\pi x a_{vx}}{L}\right)+v_y\sin\left(\frac{\pi y a_{vy}}{L}\right)+v_z\sin\left(\frac{\pi z a_{vz}}{L}\right)+v_0\right]+\\
&-\frac{\gamma}{\gamma-1}\frac{\pi a_{pz} p_z}{L}\sin\left(\frac{\pi z a_{pz}}{L}\right)\left[w_x\sin\left(\frac{\pi x a_{wx}}{L}\right)+w_y\sin\left(\frac{\pi y a_{wy}}{L}\right)+w_z\cos\left(\frac{\pi z a_{wz}}{L}\right)+w_0\right]+\\
&+\frac{\pi a_{px}\rho_x}{2L}\cos\left(\frac{\pi x a_{px}}{L}\right)\left[u_x\sin\left(\frac{\pi x a_{ux}}{L}\right)+u_y\cos\left(\frac{\pi y a_{uy}}{L}\right)+u_z\cos\left(\frac{\pi z a_{uz}}{L}\right)+u_0\right]^2\left(\left[u_x\sin\left(\frac{\pi x a_{ux}}{L}\right)+u_y\cos\left(\frac{\pi y a_{uy}}{L}\right)+u_z\cos\left(\frac{\pi z a_{uz}}{L}\right)+u_0\right]^2\right.\\
&\quad\left.+\left[v_x\cos\left(\frac{\pi x a_{vx}}{L}\right)+v_y\sin\left(\frac{\pi y a_{vy}}{L}\right)+v_z\sin\left(\frac{\pi z a_{vz}}{L}\right)+v_0\right]^2+\left[w_x\sin\left(\frac{\pi x a_{wx}}{L}\right)+w_y\sin\left(\frac{\pi y a_{wy}}{L}\right)+w_z\cos\left(\frac{\pi z a_{wz}}{L}\right)+w_0\right]^2\right)+\\
&-\frac{\pi a_{py} p_y}{2L}\sin\left(\frac{\pi y a_{py}}{L}\right)\left[v_x\cos\left(\frac{\pi x a_{vx}}{L}\right)+v_y\sin\left(\frac{\pi y a_{vy}}{L}\right)+v_z\sin\left(\frac{\pi z a_{vz}}{L}\right)+v_0\right]\left(\left[u_x\sin\left(\frac{\pi x a_{ux}}{L}\right)+u_y\cos\left(\frac{\pi y a_{uy}}{L}\right)+u_z\cos\left(\frac{\pi z a_{uz}}{L}\right)+u_0\right]^2\right.\\
&\quad\left.+\left[v_x\cos\left(\frac{\pi x a_{vx}}{L}\right)+v_y\sin\left(\frac{\pi y a_{vy}}{L}\right)+v_z\sin\left(\frac{\pi z a_{vz}}{L}\right)+v_0\right]^2+\left[w_x\sin\left(\frac{\pi x a_{wx}}{L}\right)+w_y\sin\left(\frac{\pi y a_{wy}}{L}\right)+w_z\cos\left(\frac{\pi z a_{wz}}{L}\right)+w_0\right]^2\right)+\\
&+\frac{\pi a_{pz}\rho_z}{2L}\cos\left(\frac{\pi z a_{pz}}{L}\right)\left[w_x\sin\left(\frac{\pi x a_{wx}}{L}\right)+w_y\sin\left(\frac{\pi y a_{wy}}{L}\right)+w_z\cos\left(\frac{\pi z a_{wz}}{L}\right)+w_0\right]\left(\left[u_x\sin\left(\frac{\pi x a_{ux}}{L}\right)+u_y\cos\left(\frac{\pi y a_{uy}}{L}\right)+u_z\cos\left(\frac{\pi z a_{uz}}{L}\right)+u_0\right]^2\right.\\
&\quad\left.+\left[v_x\cos\left(\frac{\pi x a_{vx}}{L}\right)+v_y\sin\left(\frac{\pi y a_{vy}}{L}\right)+v_z\sin\left(\frac{\pi z a_{vz}}{L}\right)+v_0\right]^2+\left[w_x\sin\left(\frac{\pi x a_{wx}}{L}\right)+w_y\sin\left(\frac{\pi y a_{wy}}{L}\right)+w_z\cos\left(\frac{\pi z a_{wz}}{L}\right)+w_0\right]^2\right)+\\
&+\frac{\pi a_{ux} u_x}{2L}\cos\left(\frac{\pi x a_{ux}}{L}\right)\left\{\left(3\left[u_x\sin\left(\frac{\pi x a_{ux}}{L}\right)+u_y\cos\left(\frac{\pi y a_{uy}}{L}\right)+u_z\cos\left(\frac{\pi z a_{uz}}{L}\right)+u_0\right]^2+\left[v_x\cos\left(\frac{\pi x a_{vx}}{L}\right)+v_y\sin\left(\frac{\pi y a_{vy}}{L}\right)+v_z\sin\left(\frac{\pi z a_{vz}}{L}\right)+v_0\right]^2+\right.\right.\\
&\quad+\left[w_x\sin\left(\frac{\pi x a_{wx}}{L}\right)+w_y\sin\left(\frac{\pi y a_{wy}}{L}\right)+w_z\cos\left(\frac{\pi z a_{wz}}{L}\right)+w_0\right]^2\left)\left[\rho_x\sin\left(\frac{\pi x a_{px}}{L}\right)+\rho_y\cos\left(\frac{\pi y a_{py}}{L}\right)+\rho_z\sin\left(\frac{\pi z a_{pz}}{L}\right)+\rho_0\right]\right.\\
&\quad\left.\left.+\left[p_x\cos\left(\frac{\pi x a_{px}}{L}\right)+p_y\sin\left(\frac{\pi y a_{py}}{L}\right)+p_z\cos\left(\frac{\pi z a_{pz}}{L}\right)+p_0\right]\frac{2\gamma}{\gamma-1}\right\}+\\
&-\frac{\pi a_{uy} u_y}{L}\sin\left(\frac{\pi y a_{uy}}{L}\right)\left[v_x\cos\left(\frac{\pi x a_{vx}}{L}\right)+v_y\sin\left(\frac{\pi y a_{vy}}{L}\right)+v_z\sin\left(\frac{\pi z a_{vz}}{L}\right)+v_0\right]\left[u_x\sin\left(\frac{\pi x a_{ux}}{L}\right)+u_y\cos\left(\frac{\pi y a_{uy}}{L}\right)+u_z\cos\left(\frac{\pi z a_{uz}}{L}\right)+u_0\right]\cdot\\
&\quad\cdot\left[\rho_x\sin\left(\frac{\pi x a_{px}}{L}\right)+\rho_y\cos\left(\frac{\pi y a_{py}}{L}\right)+\rho_z\sin\left(\frac{\pi z a_{pz}}{L}\right)+\rho_0\right]+\\
&-\frac{\pi a_{uz} u_z}{L}\sin\left(\frac{\pi z a_{uz}}{L}\right)\left[w_x\sin\left(\frac{\pi x a_{wx}}{L}\right)+w_y\sin\left(\frac{\pi y a_{wy}}{L}\right)+w_z\cos\left(\frac{\pi z a_{wz}}{L}\right)+w_0\right]\left[u_x\sin\left(\frac{\pi x a_{ux}}{L}\right)+u_y\cos\left(\frac{\pi y a_{uy}}{L}\right)+u_z\cos\left(\frac{\pi z a_{uz}}{L}\right)+u_0\right]\cdot\\
&\quad\cdot\left[\rho_x\sin\left(\frac{\pi x a_{px}}{L}\right)+\rho_y\cos\left(\frac{\pi y a_{py}}{L}\right)+\rho_z\sin\left(\frac{\pi z a_{pz}}{L}\right)+\rho_0\right]+\\
&-\frac{\pi a_{vx} v_x}{L}\sin\left(\frac{\pi x a_{vx}}{L}\right)\left[v_x\cos\left(\frac{\pi x a_{vx}}{L}\right)+v_y\sin\left(\frac{\pi y a_{vy}}{L}\right)+v_z\sin\left(\frac{\pi z a_{vz}}{L}\right)+v_0\right]\left[u_x\sin\left(\frac{\pi x a_{ux}}{L}\right)+u_y\cos\left(\frac{\pi y a_{uy}}{L}\right)+u_z\cos\left(\frac{\pi z a_{uz}}{L}\right)+u_0\right]\cdot\\
&\quad\cdot\left[\rho_x\sin\left(\frac{\pi x a_{px}}{L}\right)+\rho_y\cos\left(\frac{\pi y a_{py}}{L}\right)+\rho_z\sin\left(\frac{\pi z a_{pz}}{L}\right)+\rho_0\right]+\\
&+\frac{\pi a_{vy} v_y}{2L}\cos\left(\frac{\pi y a_{vy}}{L}\right)\left\{\left(\left[u_x\sin\left(\frac{\pi x a_{ux}}{L}\right)+u_y\cos\left(\frac{\pi y a_{uy}}{L}\right)+u_z\cos\left(\frac{\pi z a_{uz}}{L}\right)+u_0\right]^2+3\left[v_x\cos\left(\frac{\pi x a_{vx}}{L}\right)+v_y\sin\left(\frac{\pi y a_{vy}}{L}\right)+v_z\sin\left(\frac{\pi z a_{vz}}{L}\right)+v_0\right]^2+\right.\right.\\
&\quad+\left[w_x\sin\left(\frac{\pi x a_{wx}}{L}\right)+w_y\sin\left(\frac{\pi y a_{wy}}{L}\right)+w_z\cos\left(\frac{\pi z a_{wz}}{L}\right)+w_0\right]^2\left)\left[\rho_x\sin\left(\frac{\pi x a_{px}}{L}\right)+\rho_y\cos\left(\frac{\pi y a_{py}}{L}\right)+\rho_z\sin\left(\frac{\pi z a_{pz}}{L}\right)+\rho_0\right]+\right.\\
&\quad\left.\left.+\left[p_x\cos\left(\frac{\pi x a_{px}}{L}\right)+p_y\sin\left(\frac{\pi y a_{py}}{L}\right)+p_z\cos\left(\frac{\pi z a_{pz}}{L}\right)+p_0\right]\frac{2\gamma}{\gamma-1}\right\}+\\
&+\frac{\pi a_{vz} v_z}{L}\cos\left(\frac{\pi z a_{vz}}{L}\right)\left[w_x\sin\left(\frac{\pi x a_{wx}}{L}\right)+w_y\sin\left(\frac{\pi y a_{wy}}{L}\right)+w_z\cos\left(\frac{\pi z a_{wz}}{L}\right)+w_0\right]\left[v_x\cos\left(\frac{\pi x a_{vx}}{L}\right)+v_y\sin\left(\frac{\pi y a_{vy}}{L}\right)+v_z\sin\left(\frac{\pi z a_{vz}}{L}\right)+v_0\right]\cdot\\
&\quad\cdot\left[\rho_x\sin\left(\frac{\pi x a_{px}}{L}\right)+\rho_y\cos\left(\frac{\pi y a_{py}}{L}\right)+\rho_z\sin\left(\frac{\pi z a_{pz}}{L}\right)+\rho_0\right]+\\
&+\frac{\pi a_{wx} w_x}{L}\cos\left(\frac{\pi x a_{wx}}{L}\right)\left[w_x\sin\left(\frac{\pi x a_{wx}}{L}\right)+w_y\sin\left(\frac{\pi y a_{wy}}{L}\right)+w_z\cos\left(\frac{\pi z a_{wz}}{L}\right)+w_0\right]\left[u_x\sin\left(\frac{\pi x a_{ux}}{L}\right)+u_y\cos\left(\frac{\pi y a_{uy}}{L}\right)+u_z\cos\left(\frac{\pi z a_{uz}}{L}\right)+u_0\right]\cdot\\
&\quad\cdot\left[\rho_x\sin\left(\frac{\pi x a_{px}}{L}\right)+\rho_y\cos\left(\frac{\pi y a_{py}}{L}\right)+\rho_z\sin\left(\frac{\pi z a_{pz}}{L}\right)+\rho_0\right]+\\
&+\frac{\pi a_{wy} w_y}{L}\cos\left(\frac{\pi y a_{wy}}{L}\right)\left[w_x\sin\left(\frac{\pi x a_{wx}}{L}\right)+w_y\sin\left(\frac{\pi y a_{wy}}{L}\right)+w_z\cos\left(\frac{\pi z a_{wz}}{L}\right)+w_0\right]\left[v_x\cos\left(\frac{\pi x a_{vx}}{L}\right)+v_y\sin\left(\frac{\pi y a_{vy}}{L}\right)+v_z\sin\left(\frac{\pi z a_{vz}}{L}\right)+v_0\right]\cdot\\
&\quad\cdot\left[\rho_x\sin\left(\frac{\pi x a_{px}}{L}\right)+\rho_y\cos\left(\frac{\pi y a_{py}}{L}\right)+\rho_z\sin\left(\frac{\pi z a_{pz}}{L}\right)+\rho_0\right]+\\
&-\frac{\pi a_{wz} w_z}{2L}\sin\left(\frac{\pi z a_{wz}}{L}\right)\left\{\left(\left[u_x\sin\left(\frac{\pi x a_{ux}}{L}\right)+u_y\cos\left(\frac{\pi y a_{uy}}{L}\right)+u_z\cos\left(\frac{\pi z a_{uz}}{L}\right)+u_0\right]^2+\left[v_x\cos\left(\frac{\pi x a_{vx}}{L}\right)+v_y\sin\left(\frac{\pi y a_{vy}}{L}\right)+v_z\sin\left(\frac{\pi z a_{vz}}{L}\right)+v_0\right]^2+\right.\right.\\
&\quad+3\left[w_x\sin\left(\frac{\pi x a_{wx}}{L}\right)+w_y\sin\left(\frac{\pi y a_{wy}}{L}\right)+w_z\cos\left(\frac{\pi z a_{wz}}{L}\right)+w_0\right]^2\left)\left[\rho_x\sin\left(\frac{\pi x a_{px}}{L}\right)+\rho_y\cos\left(\frac{\pi y a_{py}}{L}\right)+\rho_z\sin\left(\frac{\pi z a_{pz}}{L}\right)+\rho_0\right]+\right.\\
&\quad\left.+\left[p_x\cos\left(\frac{\pi x a_{px}}{L}\right)+p_y\sin\left(\frac{\pi y a_{py}}{L}\right)+p_z\cos\left(\frac{\pi z a_{pz}}{L}\right)+p_0\right]\frac{2\gamma}{\gamma-1}\right\}
\end{aligned}
$$

- Use Maple, Mathematica, automatic differentiation
- Manufactured Analytic Solution Abstraction library: `https://manufactured-solutions.github.io/MASA/`

Manufactured Residuals

The Last Resort

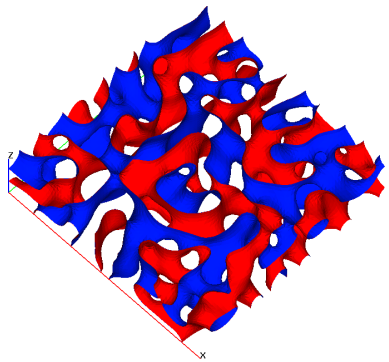
- Manufactured solution tests fail
- The bug location isn't obvious
- You can't invert the problem by hand



Manufactured Residuals

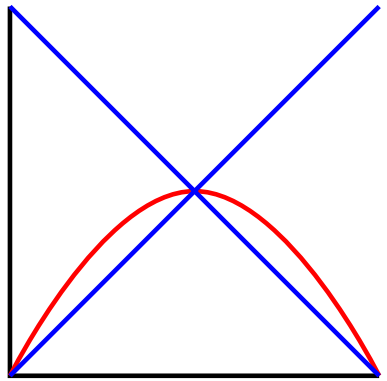
The Last Resort

- Manufactured solution tests fail
- The bug location isn't obvious
- You can't invert the problem by hand



Test $\boldsymbol{R}(\boldsymbol{U})$

- Small (one, two element!?) meshes
- Cartesian grids



Code-to-Code Comparisons

"Lumped" Verification

- Differing results from:
  - Different models
  - Different formulations
  - Different discretizations

Hierarchic Models

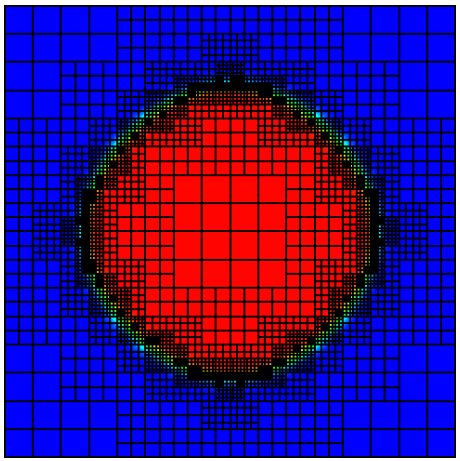
Per-Operator Tesing

- Coefficients selectively “turn off” parts of equations
- Allows easier construction of analytic solutions
- Assists with “narrowing down” other bugs

Model Simplification

- Model linearity can be tested in solver
- Reduce complex physics to simple physics case
- Code-to-code testing

Symmetry Tests



Symmetry In, Symmetry Out

- Mirror, radial symmetries
- Beware unstable solution modes!

Jacobian Verification

Inexact Newton Step

$$\begin{aligned} \boldsymbol{J}(\boldsymbol{u}^{n-1})\left(\boldsymbol{u}^{n}-\boldsymbol{u}^{n-1}\right) &\equiv -\boldsymbol{R}(\boldsymbol{u}^{n-1}) \\ \boldsymbol{J} &\equiv \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{u}} \end{aligned}$$

- Library code handles inexact solve tolerances, line search, etc.
- $\boldsymbol{R}$, $\boldsymbol{J}$ are application-dependent

Library Jacobian Construction

Finite Differencing

$$\boldsymbol{J}_{ij} \approx \frac{\boldsymbol{R}_i(\boldsymbol{u} + \varepsilon \boldsymbol{e}_j) - \boldsymbol{R}_i(\boldsymbol{u} - \varepsilon \boldsymbol{e}_j)}{2\varepsilon}$$

Greedy or element-wise algorithms handle sparsity

Complex-Step Perturbations

$$\boldsymbol{J}_{ij} \approx \frac{\Im[\boldsymbol{R}_i(\boldsymbol{u} + \varepsilon \boldsymbol{e}_j \sqrt{-1})]}{\varepsilon}$$

Avoids floating point subtractive cancellation error

Automatic Differentiation

- Variable constructors seed derivatives
- Variable operations evaluate derivatives

Jacobian Verification

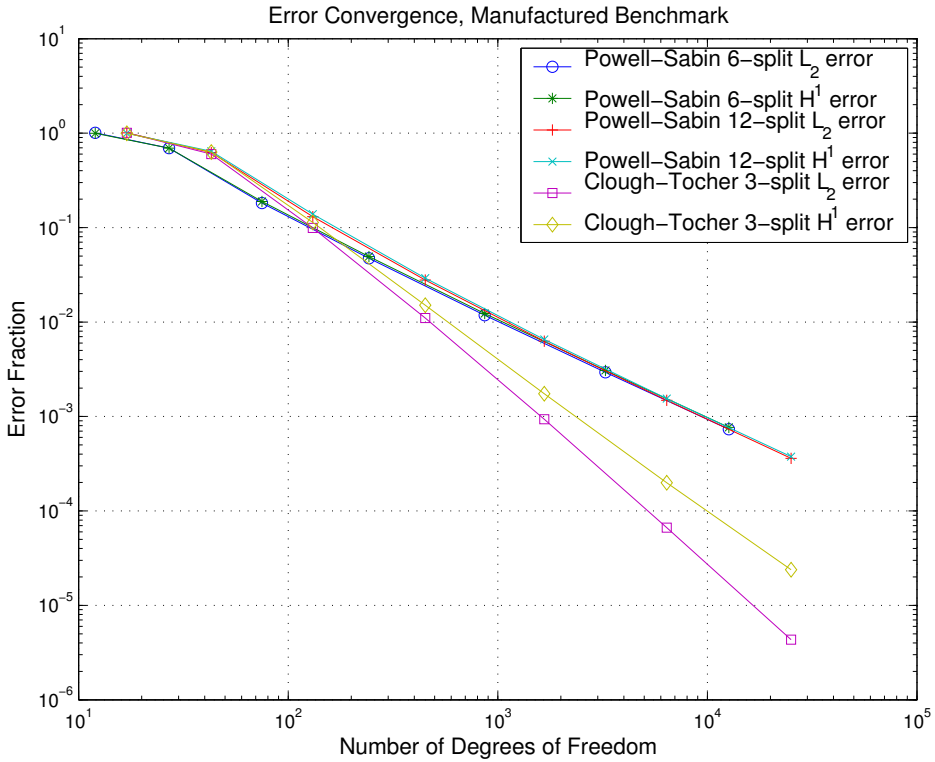
Test Analytic vs. Numeric Jacobians

- Relative error in matrix norm
- If match isn't within tolerance, either:
  - The discretization or floating point error has overwhelmed the finite differenced Jacobian
    - Unlikely for good choices of finite difference perturbations
    - Can be investigated
  - The residual is non-differentiable at that iterate
    - Can be checked analytically
  - The Jacobian calculation is wrong
  - The residual calculation is wrong

A Priori Asymptotic Convergence Rates

Biharmonic Problem, Manufactured Solution

- $\Delta^2 \boldsymbol{u} = \boldsymbol{f}$
- C^1 Macroelement bases

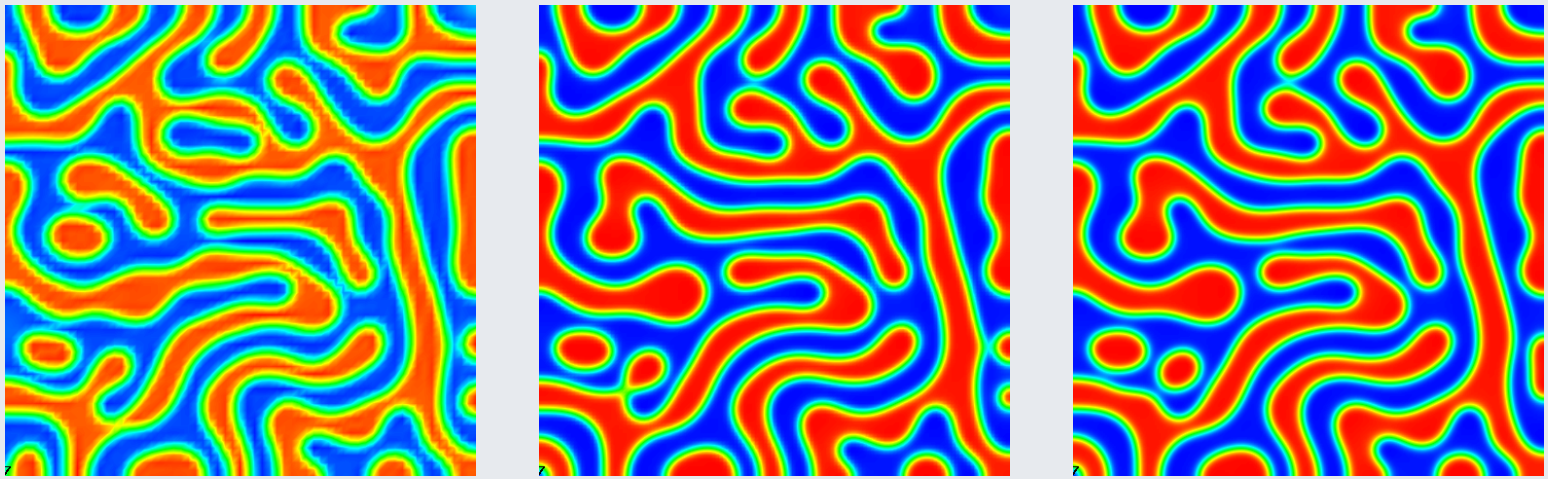


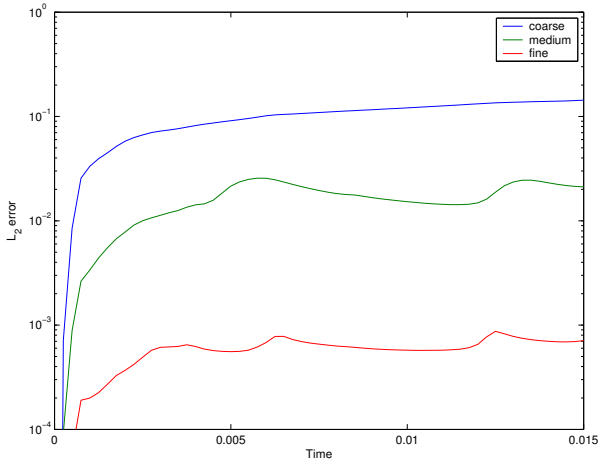
Verification Example

- Code verification failures - bugs in basis transformations
- Solution verification “failure” - higher order Nitsche lift fails for L_2 error with quadratic elements for fourth order problems

Asymptotic Convergence Rate Examples

Cahn-Hilliard Phase Evolution





Gives some confidence in even highly nonlinear, transient, stochastic problems

Thanks to Dr. Graham F. Carey

The original development team was heavily influenced by Professor Graham F. Carey, professor of aerospace engineering and engineering mechanics at The University of Texas at Austin, director of the ICES Computational Fluid Dynamics Laboratory, and holder of the Richard B. Curran Chair in Engineering.
Many of the technologies employed in libMesh were implemented because Dr. Carey taught them to us, we went back to the lab, and immediately began coding. In a very real way, he was ultimately responsible for this library that we hope you may find useful, despite his continued insistence that "no one ever got a PhD from here for writing a code."

Acknowledgements

Recent libMesh contributors:

- David Andrs
- Paul Bauman
- Vikram Garg
- Derek Gaston
- Dmitry Karpeev
- Benjamin Kirk
- David Knezevic
- Cody Permann
- John Peterson
- Sylvain Vallaghe

Useful resources:

- libMesh: `https://libmesh.github.io/`
- MOOSE: `https://mooseframework.org/`
- FALCON: `https://github.com/idaholab/falcon`
- MASA: `https://manufactured-solutions.github.io/MASA/`
- GRINS: `https://grinsfem.github.io/`
- Akselos: `http://www.akselos.com/`