LibMesh Experience and Usage

John W. Peterson
peterson@cfdlab.ae.utexas.edu

Univ. of Texas at Austin

January 12, 2007
1 Introduction
2 Weighted Residuals
3 Poisson Equation
4 Other Examples
5 Essential BCs
6 Some Extensions
Library Structure

- Basic libraries are LibMesh’s “roots”
- Application “branches” built off the library “trunk”
We assume there is a Boundary Value Problem of the form

\[
M \frac{\partial u}{\partial t} = F(u) \quad \in \Omega \\
G(u) = 0 \quad \in \Omega \\
u = u_D \quad \in \partial \Omega_D \\
N(u) = 0 \quad \in \partial \Omega_N \\
u(x, 0) = u_0(x)
\]
Associated to the problem domain $\Omega$ is a LibMesh data structure called a Mesh

A Mesh is essentially a collection of finite elements

$\Omega^h := \bigcup_e \Omega_e$
A Generic BVP

- Associated to the problem domain $\Omega$ is a LibMesh data structure called a **Mesh**
- A **Mesh** is essentially a collection of finite elements
  \[ \Omega^h := \bigcup_{e} \Omega_e \]

- LibMesh provides some simple structured mesh generation routines as well as an interface to Triangle.
1 Introduction

2 Weighted Residuals

3 Poisson Equation

4 Other Examples

5 Essential BCs

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

\[(F(u), \nu) = 0 \quad \forall \nu \in \mathcal{V}\]

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

\[(F(u^h), \nu^h) = 0 \quad \forall \nu^h \in \mathcal{V}^h\]
The point of departure in any FE analysis which uses LibMesh is the weighted residual statement

\[(F(u), v) = 0 \quad \forall v \in \mathcal{V}\]

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

\[(F(u^h), v^h) = 0 \quad \forall v^h \in \mathcal{V}^h\]
Poisson Equation

\[-\Delta u = f \quad \in \quad \Omega\]
Poisson Equation

\[-\Delta u = f \quad \in \Omega\]

Weighted Residual Statement

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

\[-k\Delta u + \mathbf{b} \cdot \nabla u = f \quad \in \quad \Omega\]
Linear Convection-Diffusion

\[-k \Delta u + \mathbf{b} \cdot \nabla u = f \quad \in \quad \Omega\]

Weighted Residual Statement

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

\[ \nabla p - \nu \Delta u = f \]
\[ \nabla \cdot u = 0 \quad \in \quad \Omega \]
**Stokes Flow**

\[
\nabla p - \nu \Delta u = f \\
\n\nabla \cdot u = 0 \\
\in \Omega
\n\]

**Weighted Residual Statement**

\[
u := [u, p] , \quad \nu := [v, q]
\]

\[
(F(u), \nu) := \int_{\Omega} \left[ -p (\nabla \cdot \nu) + \nu \nabla u : \nabla \nu - f \cdot \nu \\
+ (\nabla \cdot u) q \right] dx + \int_{\partial \Omega_N} n \cdot (\nu \nabla u - pI) \cdot \nu \, ds
\]
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.
1. Introduction
2. Weighted Residuals
3. Poisson Equation
4. Other Examples
5. Essential BCs
6. Some Extensions
For simplicity we will focus on the weighted residual statement arising from the Poisson equation, with $\partial \Omega_N = \emptyset$, 

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

\[ 0 = \int_{\Omega^h} \left[ \nabla u^h \cdot \nabla v^h - f v^h \right] dx \quad \forall v^h \in \mathcal{V}^h \]
The integral over $\Omega^h$ . . . is written as a sum of integrals over the $N_e$ finite elements:

\[
0 = \int_{\Omega^h} \left[ \nabla u^h \cdot \nabla v^h - f v^h \right] dx \quad \forall v^h \in \mathcal{V}^h \\
= \sum_{e=1}^{N_e} \int_{\Omega^e} \left[ \nabla u^h \cdot \nabla v^h - f v^h \right] dx \quad \forall v^h \in \mathcal{V}^h
\]
An element integral will have contributions only from the global basis functions corresponding to its nodes.

We call these local basis functions $\phi_i$, $0 \leq i \leq N_s$.

$$v^h|_{\Omega_e} = \sum_{i=1}^{N_s} c_i \phi_i$$
An element integral will have contributions only from the global basis functions corresponding to its nodes.

We call these local basis functions $\phi_i$, $0 \leq i \leq N_s$.

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

$$\int_{\Omega_e} \nu^h \, dx = \sum_{i=1}^{N_s} c_i \int_{\Omega_e} \phi_i \, dx$$
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 “\( \phi_i \)” 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 “\( \phi_i \)” 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

\[ K^e U^e - F^e \]
The entries of the element stiffness matrix are the integrals

\[ K_{ij}^e := \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \, dx \]
The entries of the element stiffness matrix are the integrals

\[ K_{ij}^e := \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \, dx \]

While for the element right-hand side we have

\[ F_i^e := \int_{\Omega_e} f \phi_i \, dx \]
The entries of the element stiffness matrix are the integrals

\[ K_{ij}^e := \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \, dx \]

While for the element right-hand side we have

\[ F_{i}^e := \int_{\Omega_e} f \phi_i \, dx \]

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

\[ KU = F \]
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 \( J \).

\[
F_i^e = \int_{\Omega_e} f \phi_i dx = \int_{\hat{\Omega}_e} f(x(\xi)) \phi_i |J| d\xi
\]
The integrals are performed on a “reference” element \( \hat{\Omega}_e \)

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

\[
K_{ij}^e = \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \, dx = \int_{\hat{\Omega}_e} \hat{\nabla}_\xi \phi_j \cdot \hat{\nabla}_\xi \phi_i \, |J|d\xi
\]
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 “$\xi_q$” and weights “$w_q$”. 
The integrals on the “reference” element are approximated via numerical quadrature.

The quadrature rule has $N_q$ points “$\xi_q$” and weights “$w_q$”.

\[
F^e_i = \int_{\hat{\Omega}_e} f \phi_i |J| d\xi \\
\approx \sum_{q=1}^{N_q} f(x(\xi_q)) \phi_i(\xi_q)|J(\xi_q)|w_q
\]
The integrals on the “reference” element are approximated via numerical quadrature. The quadrature rule has \( N_q \) points “\( \xi_q \)” and weights “\( w_q \).

\[
K_{ij}^e = \int_{\Omega_e} \hat{\nabla}_\xi \phi_j \cdot \hat{\nabla}_\xi \phi_i \, |J| \, d\xi \\
\approx \sum_{q=1}^{N_q} \hat{\nabla}_\xi \phi_j(\xi_q) \cdot \hat{\nabla}_\xi \phi_i(\xi_q) \, |J(\xi_q)| \, w_q
\]
LibMesh provides the following variables at each quadrature point $q$

<table>
<thead>
<tr>
<th>Code</th>
<th>Math</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JxW[q]</td>
<td>$</td>
<td>J(\xi_q)</td>
</tr>
<tr>
<td>phi[i][q]</td>
<td>$\phi_i(\xi_q)$</td>
<td>value of $i^{th}$ shape fn.</td>
</tr>
<tr>
<td>dphi[i][q]</td>
<td>$\nabla_\xi \phi_i(\xi_q)$</td>
<td>value of $i^{th}$ shape fn. gradient</td>
</tr>
<tr>
<td>xyz[q]</td>
<td>$x(\xi_q)$</td>
<td>location of $\xi_q$ in physical space</td>
</tr>
</tbody>
</table>
The **LibMesh** representation of the matrix and rhs assembly is similar to the mathematical statements.

```plaintext
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.

\[
\begin{align*}
\text{for } (q=0; \ q<Nq; \ ++q) \\
\text{for } (i=0; \ i<Ns; \ ++i) \ { } \\
F_e(i) & = JxW[q] \cdot f(xyz[q]) \cdot \phi_i(q) \cdot |J(\xi_q)| \cdot w_q \\
\end{align*}
\]
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]);
```

\[
F^e_i = \sum_{q=1}^{N_q} f(x(\xi_q)) \phi_i(\xi_q) |J(\xi_q)| w_q
\]
The LibMesh representation of the matrix and rhs assembly is similar to the mathematical statements.

\[
\text{for (q=0; q<Nq; ++q) for (i=0; i<Ns; ++i) }
\]
\[
\quad \text{Fe(i) += JxW[q]*f(xyz[q])*phi[i][q];}
\]
\[
\text{for (j=0; j<Ns; ++j) }
\]
\[
\quad \text{Ke(i, j) += JxW[q]*(dphi[j][q]*dphi[i][q]);}
\]
\[
\text{F}^e_i = \sum_{q=1}^{N_q} f(x(\xi_q)) \phi_i(\xi_q) |J(\xi_q)| w_q
\]
The \textbf{LibMesh} representation of the matrix and rhs assembly is similar to the mathematical statements.

\begin{verbatim}
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]);
}
\end{verbatim}

\[ F_i^e = \sum_{q=1}^{N_q} f(x(\xi_q)) \phi_i(\xi_q) |J(\xi_q)| w_q \]
The LibMesh representation of the matrix and rhs assembly is similar to the mathematical statements.

\[
\text{for (q=0; q<Nq; ++q) for (i=0; i<Ns; ++i) }
\]
\[
\text{Fe(i) += JxW[q]*f(xyz[q])*phi[i][q];}
\]
\[
\text{for (j=0; j<Ns; ++j)}
\]
\[
\text{Ke(i,j) += JxW[q]*(dphi[j][q]*dphi[i][q]);}
\]
\[
K_{ij}^e = \sum_{q=1}^{Nq} \hat{\nabla}_{\xi} \phi_j(\xi_q) \cdot \hat{\nabla}_{\xi} \phi_i(\xi_q) |J(\xi_q)| w_q
\]
The **LibMesh** representation of the matrix and rhs assembly is similar to the mathematical statements.

\[
\text{for (q=0; q<Nq; ++q)}
\]

\[
\text{for (i=0; i<Ns; ++i) }
\]

\[
K_{ij}^e = \sum_{q=1}^{N_q} \nabla_{\xi} \phi_j(\xi_q) \cdot \nabla_{\xi} \phi_i(\xi_q) |J(\xi_q)| w_q
\]
The LibMesh representation of the matrix and rhs assembly is similar to the mathematical statements.

\[
K_{ij}^e = \sum_{q=1}^{N_q} \nabla_{\xi} \phi_j(\xi_q) \cdot \nabla_{\xi} \phi_i(\xi_q) |J(\xi_q)| w_q
\]
The matrix assembly routine for the linear convection-diffusion equation,

\[-k\Delta u + b \cdot \nabla u = f\]

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

    for (j=0; j<Ns; ++j)
        \(Ke(i,j) += JxW[q] \ast (k \ast (dphi[j][q] \ast dphi[i][q]))\)
        \(+ (b \ast dphi[j][q]) \ast phi[i][q]);\)
The matrix assembly routine for the linear convection-diffusion equation,

\[-k\Delta u + b \cdot \nabla u = f\]

for \((q=0; \ q<Nq; \ ++q)\)

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

    for \((j=0; \ j<Ns; \ ++j)\)
        \(Ke(i,j) \quad += \quad JxW[q]*(k*(dphi[j][q]*dphi[i][q])\)
        \(+ (b*dphi[j][q])*phi[i][q]);\)
}
The matrix assembly routine for the linear convection-diffusion equation,

\[-k\Delta u + b \cdot \nabla u = f\]

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,

\[ \nabla p - \nu \Delta u = f \]
\[ \nabla \cdot u = 0 \quad \in \quad \Omega \subset \mathbb{R}^2 \]

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

\[
\begin{bmatrix}
K^e_{u_1u_1} & K^e_{u_1u_2} & K^e_{u_1p} \\
K^e_{u_2u_1} & K^e_{u_2u_2} & K^e_{u_2p} \\
K^e_{pu_1} & K^e_{pu_2} & K^e_{pp}
\end{bmatrix}
\begin{bmatrix}
U^e_{u_1} \\
U^e_{u_2} \\
U^e_p
\end{bmatrix}
- 
\begin{bmatrix}
F^e_{u_1} \\
F^e_{u_2} \\
F^e_p
\end{bmatrix}
\]

We have an array of submatrices: \( K_e \)
For multi-variable systems like Stokes flow,

\[
\nabla p - \nu \Delta u = f \\
\n\nabla \cdot u = 0 \\
\in \Omega \subset \mathbb{R}^2
\n\]

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

\[
\begin{bmatrix}
K^{e}_{uu1} & K^{e}_{u1u2} & K^{e}_{u1p} \\
K^{e}_{u2u1} & K^{e}_{u2u2} & K^{e}_{u2p} \\
K^{e}_{pu1} & K^{e}_{pu2} & K^{e}_{pp}
\end{bmatrix}
\begin{bmatrix}
U^{e}_{u1} \\
U^{e}_{u2} \\
U^{e}_{p}
\end{bmatrix}
- 
\begin{bmatrix}
F^{e}_{u1} \\
F^{e}_{u2} \\
F^{e}_{p}
\end{bmatrix}
\]

We have an array of submatrices: \( K_e [0] [0] \)
For multi-variable systems like Stokes flow,

\[ \nabla p - \nu \Delta u = f \]
\[ \nabla \cdot u = 0 \quad \in \quad \Omega \subset \mathbb{R}^2 \]

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

\[
\begin{bmatrix}
    K^e_{u1u1} & K^e_{u1u2} & K^e_{u1p} \\
    K^e_{u2u1} & K^e_{u2u2} & K^e_{u2p} \\
    K^e_{pu1} & K^e_{pu2} & K^e_{pp}
\end{bmatrix}
\begin{bmatrix}
    U^e_{u1} \\
    U^e_{u2} \\
    U^e_p
\end{bmatrix}
- \begin{bmatrix}
    F^e_{u1} \\
    F^e_{u2} \\
    F^e_p
\end{bmatrix}
\]

We have an array of submatrices: \( Ke \) [1] [1]
For multi-variable systems like Stokes flow,

\[ \nabla p - \nu \Delta u = f \]
\[ \nabla \cdot u = 0 \]
\[ \in \quad \Omega \subset \mathbb{R}^2 \]

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

\[
\begin{bmatrix}
K_{u_1u_1}^e & K_{u_1u_2}^e & K_{u_1p}^e \\
K_{u_2u_1}^e & K_{u_2u_2}^e & K_{u_2p}^e \\
K_{pu_1}^e & K_{pu_2}^e & K_{pp}^e
\end{bmatrix}
\begin{bmatrix}
U_{u_1}^e \\
U_{u_2}^e \\
U_p^e
\end{bmatrix}
- \begin{bmatrix}
F_{u_1}^e \\
F_{u_2}^e \\
F_p^e
\end{bmatrix}
\]

We have an array of submatrices: \( Ke[2][1] \)
For multi-variable systems like Stokes flow,

\[
\begin{align*}
\nabla p - \nu \Delta u &= f \\
\n\nabla \cdot u &= 0 \\
&\quad \in \quad \Omega \subset \mathbb{R}^2
\end{align*}
\]

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

\[
\begin{bmatrix}
K^{e}_{u1u1} & K^{e}_{u1u2} & K^{e}_{u1p} \\
K^{e}_{u2u1} & K^{e}_{u2u2} & K^{e}_{u2p} \\
K^{e}_{pu1} & K^{e}_{pu2} & K^{e}_{pp}
\end{bmatrix}
\begin{bmatrix}
U^{e}_{u1} \\
U^{e}_{u2} \\
U^{e}_{p}
\end{bmatrix}
- 
\begin{bmatrix}
F^{e}_{u1} \\
F^{e}_{u2} \\
F^{e}_{p}
\end{bmatrix}
\]

And an array of right-hand sides: \( F^{e} \).
For multi-variable systems like Stokes flow,\
\[
\nabla p - \nu \Delta u = f \\
\n\nabla \cdot u = 0 \\
\]
\[\in \Omega \subset \mathbb{R}^2\]

The element stiffness matrix concept can be extended to include sub-matrices
\[
\begin{bmatrix}
K^e_{u_1 u_1} & K^e_{u_1 u_2} & K^e_{u_1 p} \\
K^e_{u_2 u_1} & K^e_{u_2 u_2} & K^e_{u_2 p} \\
K^e_{pu_1} & K^e_{pu_2} & K^e_{pp}
\end{bmatrix}
\begin{bmatrix}
U^e_{u_1} \\
U^e_{u_2} \\
U^e_p
\end{bmatrix}
- \begin{bmatrix}
F^e_{u_1} \\
F^e_{u_2} \\
F^e_p
\end{bmatrix}
\]

And an array of right-hand sides: \(F_e \mathbb{0}\).
For multi-variable systems like Stokes flow,

\[
\nabla p - \nu \Delta u = f \\
\n\n\nabla \cdot u = 0 \\
\n\notin \Omega \subset \mathbb{R}^2
\]

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

\[
\begin{bmatrix}
K_{e}^{u_1 u_1} & K_{e}^{u_1 u_2} & K_{e}^{u_1 p} \\
K_{e}^{u_2 u_1} & K_{e}^{u_2 u_2} & K_{e}^{u_2 p} \\
K_{e}^{pu_1} & K_{e}^{pu_2} & K_{e}^{pp}
\end{bmatrix}
\begin{bmatrix}
U_{e}^{u_1} \\
U_{e}^{u_2} \\
U_{e}^{p}
\end{bmatrix}
- 
\begin{bmatrix}
F_{e}^{u_1} \\
F_{e}^{u_2} \\
F_{e}^{p}
\end{bmatrix}
\]

And an array of right-hand sides: \( F_e [1] \).
The matrix assembly can proceed in essentially the same way.

For the momentum equations:

```c
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]);
        }
```
Dirichlet boundary conditions can be enforced after the global stiffness matrix $K$ has been assembled.

This usually involves:

1. placing a “1” on the main diagonal of the global stiffness matrix
2. zeroing out the row entries
3. placing the Dirichlet value in the rhs vector
4. subtracting off the column entries from the rhs
Dirichlet boundary conditions can be enforced after the global stiffness matrix $K$ has been assembled.

This usually involves:

1. placing a “1” on the main diagonal of the global stiffness matrix
2. zeroing out the row entries
3. placing the Dirichlet value in the rhs vector
4. subtracting off the column entries from the rhs
Dirichlet boundary conditions can be enforced after the global stiffness matrix $K$ has been assembled.

This usually involves:

1. Placing a “1” on the main diagonal of the global stiffness matrix.
2. Zeroing out the row entries.
3. Placing the Dirichlet value in the rhs vector.
4. Subtracting off the column entries from the rhs.
Dirichlet boundary conditions can be enforced after the global stiffness matrix $K$ has been assembled. This usually involves:

1. placing a “1” on the main diagonal of the global stiffness matrix,
2. zeroing out the row entries,
3. placing the Dirichlet value in the rhs vector,
4. subtracting off the column entries from the rhs.
Dirichlet boundary conditions can be enforced after the global stiffness matrix $K$ has been assembled. This usually involves:

1. placing a “1” on the main diagonal of the global stiffness matrix
2. zeroing out the row entries
3. placing the Dirichlet value in the rhs vector
4. subtracting off the column entries from the rhs

\[
\begin{bmatrix}
k_{11} & k_{12} & k_{13} & . & . & . \\
k_{21} & k_{22} & k_{23} & . & . & . \\
k_{31} & k_{32} & k_{33} & . & . & . \\
. & . & . & . & . & . \\
\end{bmatrix},
\begin{bmatrix}
f_1 \\
f_2 \\
f_3 \\
. \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
1 & 0 & 0 & 0 & . & . & . \\
0 & k_{22} & k_{23} & . & . & . \\
0 & k_{32} & k_{33} & . & . & . \\
. & . & . & . & . & . \\
\end{bmatrix},
\begin{bmatrix}
g_1 \\
. \\
. \\
. \\
\end{bmatrix}
\begin{bmatrix}
f_2 - k_{21}g_1 \\
f_3 - k_{31}g_1 \\
. \\
. \\
\end{bmatrix}
\]
Cons of this approach:

- Works for an interpolatory finite element basis but not in general.
- May be inefficient to change individual entries once the global matrix is assembled.
- Need to enforce boundary conditions for a generic finite element basis at the element stiffness matrix level.
Cons of this approach:

- Works for an interpolatory finite element basis but not in general.
- May be inefficient to change individual entries once the global matrix is assembled.
- Need to enforce boundary conditions for a generic finite element basis at the element stiffness matrix level.
Cons of this approach:

- Works for an interpolatory finite element basis but not in general.
- May be inefficient to change individual entries once the global matrix is assembled.

Need to enforce boundary conditions for a generic finite element basis *at the element stiffness matrix level*. 
Cons of this approach:

- Works for an interpolary finite element basis but not in general.
- May be inefficient to change individual entries once the global matrix is assembled.
- Need to enforce boundary conditions for a generic finite element basis at the element stiffness matrix level.
One solution is the “penalty” boundary formulation

A term is added to the standard weighted residual statement

\[(F(u), v) + \frac{1}{\epsilon} \int_{\partial \Omega_D} (u - u_D)v \, dx = 0 \quad \forall v \in \mathcal{V}\]

penalty term

Here \(\epsilon \ll 1\) is chosen so that, in floating point arithmetic, \(\frac{1}{\epsilon} + 1 = \frac{1}{\epsilon}\).

This weakly enforces \(u = u_D\) on the Dirichlet boundary, and works for general finite element bases.
One solution is the “penalty” boundary formulation

A term is added to the standard weighted residual statement

\[ (F(u), v) + \frac{1}{\epsilon} \int_{\partial \Omega_D} (u - u_D)v \, dx = 0 \quad \forall v \in V \]

Here \( \epsilon \ll 1 \) is chosen so that, in floating point arithmetic, \( \frac{1}{\epsilon} + 1 = \frac{1}{\epsilon} \).

This weakly enforces \( u = u_D \) on the Dirichlet boundary, and works for general finite element bases.
One solution is the “penalty” boundary formulation

A term is added to the standard weighted residual statement

\[(F(u), v) + \frac{1}{\epsilon} \int_{\partial\Omega_D} (u - u_D)v \, dx = 0 \quad \forall v \in V\]

Here \(\epsilon \ll 1\) is chosen so that, in floating point arithmetic, \(\frac{1}{\epsilon} + 1 = \frac{1}{\epsilon}\).

This weakly enforces \(u = u_D\) on the Dirichlet boundary, and works for general finite element bases.
One solution is the “penalty” boundary formulation

A term is added to the standard weighted residual statement

\[(F(u), v) + \frac{1}{\epsilon} \int_{\partial\Omega_D} (u - u_D)v \, dx = 0 \quad \forall v \in V\]

Here \(\epsilon \ll 1\) is chosen so that, in floating point arithmetic, \(\frac{1}{\epsilon} + 1 = \frac{1}{\epsilon}\).

This weakly enforces \(u = u_D\) on the Dirichlet boundary, and works for general finite element bases.
Penalty Formulation

LibMesh provides:
- A quadrature rule with $N_q f$ points and $J \times W_f[]$
- A finite element coincident with the boundary face that has shape function values $\phi_f[][]$

```c
def (qf=0; qf<Nqf; ++qf) {
    for (i=0; i<Nf; ++i) {
        Fe(i) += JxW_f[qf]*
            penalty*uD(xyz[q])*phi_f[i][qf];
    }
    for (j=0; j<Nf; ++j) {
        Ke(i,j) += JxW_f[qf]*
            penalty*phi_f[j][qf]*phi_f[i][qf];
    }
}
```
LibMesh provides:

- A quadrature rule with $N_{qf}$ points and $JxW_f[]$
- A finite element coincident with the boundary face that has shape function values $\phi_{f[]}[]$

```cpp
for (qf=0; qf<Nqf; ++qf) {
    for (i=0; i<Nf; ++i) {
        Fe(i) += JxW_f[qf]*
            penalty*uD(xyz[q]) * phi_f[i][qf];
    }
    for (j=0; j<Nf; ++j)
        Ke(i,j) += JxW_f[qf]*
            penalty*phi_f[j][qf]*phi_f[i][qf];
}
```
LibMesh provides:
- A quadrature rule with $N_{qf}$ points and $J_x W_{f[\cdot]}$
- A finite element coincident with the boundary face that has shape function values $\phi_{i,f[\cdot][\cdot]}$

```c
for (qf=0; qf<Nqf; ++qf) {
    for (i=0; i<Nf; ++i) {
        Fe(i) += JxW_f[qf] * penalty*uD(xyz[q])*phi_f[i][qf];
    }
    for (j=0; j<Nf; ++j) {
        Ke(i,j) += JxW_f[qf] * penalty*phi_f[j][qf]*phi_f[i][qf];
    }
}
```
1 Introduction
2 Weighted Residuals
3 Poisson Equation
4 Other Examples
5 Essential BCs
6 Some Extensions
For linear problems, we have already seen how the weighted residual statement leads directly to a sparse linear system of equations

\[ KU = F \]
For time-dependent problems,

\[
\frac{\partial u}{\partial t} = F(u)
\]

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

\[
\left( \frac{u^{n+1} - u^n}{\Delta t}, v^h \right) = (F(u_{\theta}), v^h) \quad \forall v^h \in V^h
\]

\[
u_{\theta} := \theta u^{n+1} + (1 - \theta) u^n
\]

Leads to $KU = F$ at each timestep.
For nonlinear problems, typically a sequence of linear problems must be solved, e.g. for Newton’s method

\[(F'(u^k)\delta u^{k+1}, v) = -(F(u^k), v)\]

where \(F'(u^k)\) is the linearized (Jacobian) operator associated with the PDE.

Must solve \(KU = F\) (Inexact Newton method) at each iteration step.
Common to each is the need to solve a linear system (or systems) of equations \( KU = F \).

LibMesh provides several of the tools necessary to construct these systems, but it is not specifically written to solve any one problem.

We now briefly show a few of the non-trivial applications which have been built on top of the library.
Tetrahedral mesh of “pipe” geometry. Stream ribbons colored by temperature.
Adaptive grid solution shown with temperature contours and velocity vectors.
Solute contours: a plume of warm, low-salinity fluid is convected upward through a porous medium.
The tumor secretes a chemical which stimulates blood vessel formation.
Tetrahedral mesh of “pipe” geometry. Stream ribbons colored by temperature.
Adaptive grid solution shown with temperature contours and velocity vectors.
Solute contours: a plume of warm, low-salinity fluid is convected upward through a porous medium.
The tumor secretes a chemical which stimulates blood vessel formation.