Finite Element Approximation of Partial Differential Equations Using FreeFem++

or: How I Learned to Stop Worrying and Love Numerical Analysis

John C. Chrispell and Jason S. Howell

Department of Mathematical Sciences
Clemson University

February 13, 2007
SIAM Student Chapter Seminar
Department of Mathematics
University of South Carolina
Columbia, SC
Outline

1 Partial Differential Equations
   - Introduction
   - Some Example PDEs

2 How can we go about approximating PDEs?
   - Example: Poisson Problem
   - Discrete Formulation

3 About FreeFem++
   - FreeFem++ Description
   - General Program Structure

4 Sample FreeFem++ Programs
   - Poisson Problem
   - Stokes Problem

5 Advanced Topics

6 Concluding Remarks
Outline

1. Partial Differential Equations
   - Introduction
   - Some Example PDEs

2. How can we go about approximating PDEs?
   - Example: Poisson Problem
   - Discrete Formulation

3. About FreeFem++
   - FreeFem++ Description
   - General Program Structure

4. Sample FreeFem++ Programs
   - Poisson Problem
   - Stokes Problem

5. Advanced Topics

6. Concluding Remarks
What is a PDE?

**Answer:** A system of unknown functions involving

- Two or more independent variables
- Derivatives with respect to the independent variables
- Typically used to model a physical phenomenon
- Systems may include initial and/or boundary conditions
What is a PDE?

**Answer:** A system of unknown functions involving

- Two or more independent variables
- Derivatives with respect to the independent variables
- Typically used to model a physical phenomenon
- Systems may include initial and/or boundary conditions
What is a PDE?

**Answer:** A system of unknown functions involving

- Two or more independent variables
- Derivatives with respect to the independent variables
- Typically used to model a physical phenomenon
- Systems may include initial and/or boundary conditions
What is a PDE?

**Answer:** A system of unknown functions involving
- Two or more independent variables
- Derivatives with respect to the independent variables
- Typically used to model a physical phenomenon
- Systems may include initial and/or boundary conditions
Outline

1 Partial Differential Equations
   - Introduction
   - Some Example PDEs

2 How can we go about approximating PDEs?
   - Example: Poisson Problem
   - Discrete Formulation

3 About FreeFem++
   - FreeFem++ Description
   - General Program Structure

4 Sample FreeFem++ Programs
   - Poisson Problem
   - Stokes Problem

5 Advanced Topics

6 Concluding Remarks

Finite Element Approximation of Partial Differential Equations Using FreeFem++

Chrispell and Howell

CLEMSON UNIVERSITY

Mathematical Sciences
Laplace’s Equation

Find \( u \) such that:

\[
\Delta u = 0, \quad x \text{ in } \Omega \\
u(x) = g, \quad x \text{ on } \partial \Omega
\]

Used in steady state fluid flow, heat flow, or electrostatics (models diffusion).

Notation:

\[
\Omega \subset \mathbb{R}^d, \text{ for } d \in \{1, 2, 3\} \\
\partial \Omega = \Gamma = \text{Boundary of } \Omega \\
\nabla = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \end{bmatrix}^T \\
\Delta = \nabla \cdot \nabla
\]
Laplace’s Equation

Find $u$ such that:

$$\Delta u = 0, \ x \text{ in } \Omega$$

$$u(x) = g, \ x \text{ on } \partial \Omega$$

Used in steady state fluid flow, heat flow, or electrostatics (models diffusion).

Notation:

$$\Omega \subset \mathbb{R}^d, \text{ for } d \in \{1, 2, 3\}$$

$$\partial \Omega = \Gamma = \text{ Boundary of } \Omega$$

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \end{bmatrix}^T$$

$$\Delta = \nabla \cdot \nabla$$
Convection-Diffusion Problem

Find $u$ such that:

$$-\Delta u + b \cdot \nabla u + cu = f, \ x \in \Omega$$
$$u = g, \ x \text{ on } \partial \Omega.$$ 

- Added a convection term with a velocity field $b$
- Two source/sink terms: $cu$ and $f$
- $u$ models the concentration of a particle/substance over $\Omega$
Convection-Diffusion Problem

Find $u$ such that:

\[-\Delta u + b \cdot \nabla u + cu = f, \quad x \in \Omega\]

\[u = g, \quad x \text{ on } \partial \Omega.\]

- Added a convection term with a velocity field $b$
- Two source/sink terms: $cu$ and $f$
- $u$ models the concentration of a particle/substance over $\Omega$
Convection-Diffusion Problem

Find $u$ such that:

$$-\Delta u + b \cdot \nabla u + cu = f, \quad x \in \Omega$$

$$u = g, \quad x \text{ on } \partial \Omega.$$

- Added a convection term with a velocity field $b$
- Two source/sink terms: $cu$ and $f$
- $u$ models the concentration of a particle/substance over $\Omega$
Stokes Problem

Find $u$ and $p$ such that:

\[-\Delta u + \nabla p = f, \quad x \in \Omega\]
\[\nabla \cdot u = 0, \quad x \in \Omega\]
\[u = g, \quad x \text{ on } \partial \Omega\]

- Models the steady state flow of viscous fluid
- $u$ denotes fluid velocity
- $p$ denotes pressure
- Conservation of mass: $\nabla \cdot u$ (“incompressibility condition”)
Stokes Problem

Find \( \mathbf{u} \) and \( p \) such that:

\[-\Delta \mathbf{u} + \nabla p = f, \quad x \in \Omega\]
\[\nabla \cdot \mathbf{u} = 0, \quad x \in \Omega\]
\[\mathbf{u} = g, \quad x \text{ on } \partial \Omega\]

- Models the steady state flow of viscous fluid
- \( \mathbf{u} \) denotes fluid velocity
- \( p \) denotes pressure
- Conservation of mass: \( \nabla \cdot \mathbf{u} \) ("incompressibility condition")
Stokes Problem

Find $u$ and $p$ such that:

$$- \Delta u + \nabla p = f, \quad x \in \Omega$$

$$\nabla \cdot u = 0, \quad x \in \Omega$$

$$u = g, \quad x \text{ on } \partial \Omega$$

- Models the steady state flow of viscous fluid
- $u$ denotes fluid velocity
- $p$ denotes pressure
- Conservation of mass: $\nabla \cdot u$ ("incompressibility condition")
Stokes Problem

Find \( u \) and \( p \) such that:

\[
-\Delta u + \nabla p = f, \quad x \in \Omega \\
\nabla \cdot u = 0, \quad x \in \Omega \\
u = g, \quad x \text{ on } \partial\Omega
\]

- Models the steady state flow of viscous fluid
- \( u \) denotes fluid velocity
- \( p \) denotes pressure
- Conservation of mass: \( \nabla \cdot u \) ("incompressibility condition")
Navier-Stokes Equations

Find $\mathbf{u}$, and $p$ such that

$$
Re \left( \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) - \Delta \mathbf{u} + \nabla p = f, \ x \in \Omega
$$

$$
\nabla \cdot \mathbf{u} = 0, \ x \in \Omega
$$

$$
\mathbf{u} = g, \ x \text{ on } \partial \Omega
$$

- Models the flow of a viscous, incompressible, Newtonian fluid
- Problem is time-dependent
- The $\mathbf{u} \cdot \nabla \mathbf{u}$ advection (transport) term makes the problem nonlinear
Navier-Stokes Equations

Find $u$, and $p$ such that

$$Re \left( \frac{\partial u}{\partial t} + u \cdot \nabla u \right) - \Delta u + \nabla p = f, \quad x \in \Omega$$

$$\nabla \cdot u = 0, \quad x \in \Omega$$

$$u = g, \quad x \text{ on } \partial \Omega$$

- Models the flow of a viscous, incompressible, Newtonian fluid
- Problem is time-dependent
- The $u \cdot \nabla u$ advection (transport) term makes the problem nonlinear
Navier-Stokes Equations

Find $u$, and $p$ such that

$$Re \left( \frac{\partial u}{\partial t} + u \cdot \nabla u \right) - \Delta u + \nabla p = f, \ x \in \Omega$$

$$\nabla \cdot u = 0, \ x \in \Omega$$

$$u = g, \ x \text{ on } \partial \Omega$$

- Models the flow of a viscous, incompressible, Newtonian fluid
- Problem is time-dependent
- The $u \cdot \nabla u$ advection (transport) term makes the problem nonlinear
Outline

1 Partial Differential Equations
   - Introduction
   - Some Example PDEs

2 How can we go about approximating PDEs?
   - Example: Poisson Problem
   - Discrete Formulation

3 About FreeFem++
   - FreeFem++ Description
   - General Program Structure

4 Sample FreeFem++ Programs
   - Poisson Problem
   - Stokes Problem

5 Advanced Topics

6 Concluding Remarks
Notation

Function Spaces:

\[ L^2(\Omega) = \left\{ v \in \Omega : \int_{\Omega} v^2 \, d\Omega < \infty \right\} \quad (1) \]

\[ H^1(\Omega) = \left\{ v \in L^2(\Omega) : \nabla u \in L^2(\Omega) \right\} \quad (2) \]

\[ V = H^1_0(\Omega) = \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega \right\} \quad (3) \]

Respective inner products and norms:

\[ L^2(\Omega) : \| f \|_0 = (f, f)^{1/2} \]

where \((f, g) = \int_{\Omega} fg \, d\Omega\)

\[ H^1(\Omega) : \| f \|_1 = ((f, f) + (\nabla f, \nabla f))^{1/2} \]

with \((\nabla f, \nabla g) = \int_{\Omega} \nabla f \cdot \nabla g \, d\Omega\)
Notation

Function Spaces:

\[ L^2(\Omega) = \left\{ v \in \Omega : \int_{\Omega} v^2 \, d\Omega < \infty \right\} \]  
\[ H^1(\Omega) = \left\{ v \in L^2(\Omega) : \nabla u \in L^2(\Omega) \right\} \]  
\[ V = H^1_0(\Omega) = \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega \right\} \]

Respective inner products and norms:

\[ L^2(\Omega) : \| f \|_0 = (f, f)^{1/2} \]

where \((f, g) = \int_{\Omega} fg \, d\Omega\)

\[ H^1(\Omega) : \| f \|_1 = ((f, f) + (\nabla f, \nabla f))^{1/2} \]

with \((\nabla f, \nabla g) = \int_{\Omega} \nabla f \cdot \nabla g \, d\Omega\)
Notation

Function Spaces:

\[
L^2(\Omega) = \left\{ v \in \Omega : \int_{\Omega} v^2 \, d\Omega < \infty \right\} \quad (1)
\]

\[
H^1(\Omega) = \left\{ v \in L^2(\Omega) : \nabla u \in L^2(\Omega) \right\} \quad (2)
\]

\[
V = H^1_0(\Omega) = \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \partial \Omega \right\} \quad (3)
\]

Respective inner products and norms:

\[
L^2(\Omega) : \| f \|_0 = (f, f)^{1/2}
\]

where \((f, g) = \int_{\Omega} fg \, d\Omega\)

\[
H^1(\Omega) : \| f \|_1 = ((f, f) + (\nabla f, \nabla f))^{1/2}
\]

with \((\nabla f, \nabla g) = \int_{\Omega} \nabla f \cdot \nabla g \, d\Omega\)
The Poisson Problem

Find $u$ such that:

$$-\Delta u = f, \quad x \in \Omega \subset \mathbb{R}^2$$

$$u = 0, \quad \text{on } \partial \Omega.$$
The Poisson Problem

Find $u$ such that:

$$-\Delta u = f, \quad x \in \Omega \subset \mathbb{R}^2$$

$$u = 0, \quad \text{on } \partial \Omega.$$ 

How do we go about finding $u$?
The Poisson Problem

Find $u$ such that:

$$-\Delta u = f, \quad x \in \Omega \subset \mathbb{R}^2$$

$$u = 0, \quad \text{on } \partial \Omega.$$ 

How do we go about finding $u$?

• Start by considering a variational formulation:
The Poisson Problem

Find $u$ such that:

$$-\Delta u = f, \quad x \in \Omega \subset \mathbb{R}^2$$

$$u = 0, \quad \text{on } \partial \Omega.$$

How do we go about finding $u$?

- Start by considering a variational formulation:

Find

$$u \in V = H^1_0(\Omega) = \{v \in H^1(\Omega) : v = 0 \text{ on } \partial \Omega\}$$

such that

$$\int_{\Omega} -\Delta uv \, d\Omega = \int_{\Omega} fv \, d\Omega, \quad \forall \ v \in V.$$
The Poisson Problem

Find \( u \) such that:

\[
-\Delta u = f, \quad x \in \Omega \subset \mathbb{R}^2 \\
 u = 0, \quad \text{on } \partial \Omega.
\]

How do we go about finding \( u \)?

- Start by considering a variational formulation:

After integrating by parts:

Find \( u \in V = H^1_0(\Omega) = \{ \nu \in H^1(\Omega) : \nu = 0 \text{ on } \partial \Omega \} \)

such that \( \int_{\Omega} \nabla u \cdot \nabla v \, d\Omega = \int_{\Omega} fv \, d\Omega, \quad \forall \, \nu \in V. \)
The Poisson Problem

Find $u$ such that:

$$-\Delta u = f, \quad x \in \Omega \subset \mathbb{R}^2$$

$$u = 0, \quad \text{on } \partial \Omega.$$

How do we go about finding $u$?

- Start by considering a variational formulation:

After Integrating by parts:

Find $u \in V = H^1_0(\Omega) = \{v \in H^1(\Omega) : v = 0 \text{ on } \partial \Omega \}$ such that

$$\int_{\Omega} \nabla u \cdot \nabla v \, d\Omega = \int_{\Omega} fv \, d\Omega, \quad \forall \, v \in V.$$

- Note $u$ and $v$ are both in $V$
Approximating Spaces

How can an approximation to $u$ be found?

Idea:

- Determine an approximation space for $u$ (trial space)
- Determine an approximation space for $v$ (test space)
- Form the approximating system of algebraic equations
- Solve the system
Approximating Spaces

How can an approximation to $u$ be found?

Idea:

- Determine an approximation space for $u$ (trial space)
- Determine an approximation space for $v$ (test space)
- Form the approximating system of algebraic equations
- Solve the system
Approximating Spaces

How can an approximation to $u$ be found?

Idea:

- Determine an approximation space for $u$ (trial space)
- Determine an approximation space for $v$ (test space)
- Form the approximating system of algebraic equations
- Solve the system
Approximating Spaces

How can an approximation to $u$ be found?

Idea:

- Determine an approximation space for $u$ (trial space)
- Determine an approximation space for $v$ (test space)
- Form the approximating system of algebraic equations
- Solve the system
Triangulate $\Omega$

Working with the problem domain $\Omega$: Let $T_h$ be a triangulation of $\Omega$ $\Omega = \bigcup K, \ K \in T_h.$

Notation:

- $h_K$ is the diameter of triangle $K$
- $\mathcal{P}_k(K) = \text{polynomials on } K \text{ of degree } \leq k$
- $C(\Omega) = \text{continuous functions on } \Omega$

$V^h = \left\{ v \in V \cap C(\Omega) : \left. v \right|_K \in \mathcal{P}_k(K), \ \forall K \in T_h \right\}$
Triangulate $\Omega$

Working with the problem domain $\Omega$: Let $T_h$ be a triangulation of $\Omega$

\[ \Omega = \bigcup K, \ K \in T_h. \]

Notation:
- $h_K$ is the diameter of triangle $K$
- $P_k(K) = \text{polynomials on } K \text{ of degree } \leq k$
- $C(\Omega) = \text{continuous functions on } \Omega$

\[ V^h = \left\{ v \in V \cap C(\Omega) : \begin{array}{c} v \bigg|_K \in P_k(K), \forall K \in T_h \end{array} \right\} \]
Triangulate $\Omega$

Working with the problem domain $\Omega$:
Let $T_h$ be a triangulation of $\Omega$

$$\Omega = \bigcup K, \ K \in T_h.$$ 

Notation:

- $h_K$ is the diameter of triangle $K$
- $P_k(K) = \text{polynomials on } K \text{ of degree } \leq k$
- $C(\Omega) = \text{continuous functions on } \Omega$

$$V^h = \left\{ v \in V \cap C(\Omega) : \left. v \right|_K \in P_k(K), \forall K \in T_h \right\}$$
Triangulate $\Omega$

Working with the problem domain $\Omega$:

Let $T_h$ be a triangulation of $\Omega$

$$\Omega = \bigcup K, \ K \in T_h.$$ 

Notation:

- $h_K$ is the diameter of triangle $K$
- $\mathcal{P}_k(K)$ = polynomials on $K$ of degree $\leq k$
- $C(\Omega)$ = continuous functions on $\Omega$

$$V^h = \left\{ v \in V \cap C(\Omega) : \left. v \right|_K \in \mathcal{P}_k(K), \forall \ K \in T_h \right\}$$
Triangulate $\Omega$

Working with the problem domain $\Omega$: Let $T_h$ be a triangulation of $\Omega$

$$\Omega = \bigcup K, \ K \in T_h.$$ 

Notation:

- $h_K$ is the diameter of triangle $K$
- $\mathcal{P}_k(K) = \text{polynomials on } K \text{ of degree } \leq k$
- $C(\Omega) = \text{continuous functions on } \Omega$

$$V^h = \left\{ v \in V \cap C(\Omega) : \left. v \right|_K \in \mathcal{P}_k(K), \forall K \in T_h \right\}$$

Triangulation 1
Triangulate $\Omega$

Working with the problem domain $\Omega$: Let $T_h$ be a triangulation of $\Omega$

$$\Omega = \bigcup K, \ K \in T_h.$$

Notation:

- $h_K$ is the diameter of triangle $K$
- $P_k(K) = \text{polynomials on } K \text{ of degree } \leq k$
- $C(\Omega) = \text{continuous functions on } \Omega$

$$V^h = \left\{ v \in V \cap C(\Omega) : \right.$$\[ v \bigg|_K \in P_k(K), \ \forall \ K \in T_h \right\}$$
Triangulate $\Omega$

Working with the problem domain $\Omega$:
Let $T_h$ be a triangulation of $\Omega$

$$\Omega = \bigcup K, \; K \in T_h.$$ 

Notation:
- $h_K$ is the diameter of triangle $K$
- $\mathcal{P}_k(K)$ = polynomials on $K$ of degree $\leq k$
- $C(\Omega) = $ continuous functions on $\Omega$

$$V^h = \left\{ v \in V \cap C(\Omega) : \forall K \in T_h \; v \bigg|_K \in \mathcal{P}_k(K) \right\}$$

Triangulation 3
Outline

1 Partial Differential Equations
   - Introduction
   - Some Example PDEs

2 How can we go about approximating PDEs?
   - Example: Poisson Problem
   - Discrete Formulation

3 About FreeFem++
   - FreeFem++ Description
   - General Program Structure

4 Sample FreeFem++ Programs
   - Poisson Problem
   - Stokes Problem

5 Advanced Topics

6 Concluding Remarks
Recall the variational formulation:

Find \( u \in V = H^1_0(\Omega) = \{ v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega \} \)

such that \( \int_{\Omega} \nabla u \cdot \nabla v \, d\Omega = \int_{\Omega} f v \, d\Omega, \quad \forall \, v \in V. \)

Approximate with discrete variational formulation:

Find \( u^h \in V^h = \{ v \in V \cap C(\Omega) : v \big|_K \in \mathcal{P}_k(K), \forall \, K \in T_h \} \)

such that \( \int_{\Omega} \nabla u^h \cdot \nabla v^h \, d\Omega = \int_{\Omega} f v^h \, d\Omega, \quad \forall \, v^h \in V^h. \)

Here we choose the trial space (for \( u^h \)), and the test space (for \( v^h \)) to be \( V^h \).
Recall the variational formulation:

Find \( u \in V = H^1_0(\Omega) = \{ v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega \} \)

such that \( \int_{\Omega} \nabla u \cdot \nabla v \, d\Omega = \int_{\Omega} f v \, d\Omega, \quad \forall \, v \in V. \)

Approximate with discrete variational formulation:

Find \( u^h \in V^h = \left\{ v \in V \cap C(\Omega) : v \bigg|_K \in P_k(K), \forall \, K \in T_h \right\} \)

such that \( \int_{\Omega} \nabla u^h \cdot \nabla v^h \, d\Omega = \int_{\Omega} f v^h \, d\Omega, \quad \forall \, v^h \in V^h. \)

Here we choose the trial space (for \( u^h \)), and the test space (for \( v^h \)) to be \( V^h \)
Find a Basis for $V^h$

In one dimension on each element

$$V^h = \text{span}\{\phi_j\}, j = 1, \ldots, N$$

Linears:

$$\phi_i = \begin{cases} \frac{x-x_{i-1}}{x_i-x_{i-1}} & x \in [x_{i-1}, x_i] \\ \frac{x_{i+1}-x}{x_{i+1}-x_i} & x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise.} \end{cases}$$

or

$$V^h = \text{span}\{\phi_j\}, j = 1, \ldots, N$$

Quadratics:

$$\phi_1(\eta) = 2(\eta - 1/2)(\eta - 1)$$
$$\phi_2(\eta) = 4\eta(1 - \eta)$$
$$\phi_3(\eta) = 2\eta(\eta - 1/2)$$

Chrispell and Howell
Find a Basis for $V^h$

In one dimension on each element

$V^h = \text{span}\{\phi_j\}, j = 1, \ldots, N$

Linears:

$$
\phi_i = \begin{cases} 
\frac{x-x_{i-1}}{x_i-x_{i-1}} & x \in [x_{i-1}, x_i] \\
\frac{x_{i+1}-x}{x_{i+1}-x_i} & x \in [x_i, x_{i+1}] \\
0 & \text{otherwise.}
\end{cases}
$$

or

$V^h = \text{span}\{\phi_j\}, j = 1, \ldots, N$

Quadratics:

$$
\phi_1(\eta) = 2(\eta - 1/2)(\eta - 1) \\
\phi_2(\eta) = 4\eta(1-\eta) \\
\phi_3(\eta) = 2\eta(\eta - 1/2)
$$

Chrispell and Howell

Finite Element Approximation of Partial Differential Equations Using FreeFem++
Find a Basis for $V^h$

In one dimension on each element

$$V^h = \text{span}\{\phi_j\}, j = 1, \ldots, N$$

Linears:

$$\phi_i = \begin{cases} 
\frac{x-x_{i-1}}{x_i-x_{i-1}} & x \in [x_{i-1}, x_i] \\
\frac{x_{i+1}-x}{x_{i+1}-x_i} & x \in [x_i, x_{i+1}] \\
0 & \text{otherwise.}
\end{cases}$$

Quadratics:

$$\phi_1(\eta) = 2(\eta - 1/2)(\eta - 1)$$

$$\phi_2(\eta) = 4\eta(1-\eta)$$

$$\phi_3(\eta) = 2\eta(\eta - 1/2)$$

Linear Basis (1-D)

Quadratic Basis (1-D)
Find a Basis for $V^h$

In one dimension on each element

$$V^h = \text{span}\{\phi_j\}, j = 1, \ldots, N$$

**Linears:**

$$\phi_i = \begin{cases} \frac{x-x_{i-1}}{x_i-x_{i-1}} & x \in [x_{i-1}, x_i] \\ \frac{x_{i+1}-x}{x_{i+1}-x_i} & x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise.} \end{cases}$$

**Quadratics:**

$$\phi_1(\eta) = 2(\eta - 1/2)(\eta - 1)$$

$$\phi_2(\eta) = 4\eta(1-\eta)$$

$$\phi_3(\eta) = 2\eta(\eta - 1/2)$$

Linear Basis (1-D)

Quadratic Basis (1-D)
Find a Basis for $V^h$

In two dimensions we use “Tent Functions.” For example defined by

$$\phi_i(x, y) = \text{continuous piecewise linears on each triangle}$$

such that

$$\phi_i(x_i) = 1 \quad \text{and} \quad \phi_i(x_j) = 0 \quad \text{if } j \neq i$$

Note: All defined basis functions have local support
Find a Basis for $V^h$

In two dimensions we use “Tent Functions.” For example defined by

$$\phi_i(x, y) = \text{continuous piecewise linears on each triangle}$$

such that

$$\phi_i(x_i) = 1 \quad \text{and} \quad \phi_i(x_j) = 0 \quad \text{if} \ j \neq i$$

Note: All defined basis functions have **local support**
Assemble the Approximating System

Approximate

\[ u(x) \approx u^h(x) = \sum_{j=1}^{N} c_j \phi_j(x). \]

The approximating system with \( v^h = \phi_i(x) \) becomes:

\[
\int_{\Omega} \nabla \sum_{j=1}^{N} c_j \phi_j(x) \cdot \nabla \phi_i(x) \, d\Omega = \int_{\Omega} f \phi_i(x) \, d\Omega
\]

\[
\Rightarrow \sum_{j=1}^{N} \left[ \int_{\Omega} \nabla \phi_j(x) \cdot \nabla \phi_i(x) \, d\Omega \right] c_j = \int_{\Omega} f \phi_i(x) \, d\Omega
\]

\[
\Rightarrow \sum_{j=1}^{N} a_{ij} c_j = b_i
\]
The Approximating System

Using all the test elements $\phi_i \in V^h$ corresponding to “interior nodes” in $T_h$ we have:

\[ A c = b \]

where

\[ a_{ij} = \int_{\Omega} \nabla \phi_j(x) \cdot \nabla \phi_i(x) \, d\Omega \]

\[ b_i = \int_{\Omega} f \phi_i(x) \, d\Omega \]

Due to the local support of the basis functions $a_{ij} = 0$ unless there is a triangle that has both nodes $i$ and $j$.

- Systems are sparse
- Refining the approximation yields larger systems
The Approximating System

Using all the test elements $\phi_i \in V^h$ corresponding to “interior nodes” in $T_h$ we have:

$$Ac = b$$

where

$$a_{ij} = \int_{\Omega} \nabla \phi_j(\mathbf{x}) \cdot \nabla \phi_i(\mathbf{x}) \, d\Omega$$

$$b_i = \int_{\Omega} f \phi_i(\mathbf{x}) \, d\Omega$$

Due to the local support of the basis functions $a_{ij} = 0$ unless there is a triangle that has both nodes $i$ and $j$.

- Systems are sparse
- Refining the approximation yields larger systems
The Approximating System

Using all the test elements $\phi_i \in V^h$ corresponding to “interior nodes” in $T_h$ we have:

$$ Ac = b $$

where

$$ a_{ij} = \int_\Omega \nabla \phi_j(x) \cdot \nabla \phi_i(x) \, d\Omega $$

$$ b_i = \int_\Omega f \phi_i(x) \, d\Omega $$

Due to the local support of the basis functions $a_{ij} = 0$ unless there is a triangle that has both nodes $i$ and $j$.

- Systems are sparse
- Refining the approximation yields larger systems
The Approximating System

Using all the test elements $\phi_i \in V^h$ corresponding to “interior nodes” in $T_h$ we have:

$$Ac = b$$

where

$$a_{ij} = \int_\Omega \nabla \phi_j(x) \cdot \nabla \phi_i(x) \, d\Omega$$

$$b_i = \int_\Omega f \phi_i(x) \, d\Omega$$

Due to the local support of the basis functions $a_{ij} = 0$ unless there is a triangle that has both nodes $i$ and $j$.

- Systems are sparse
- Refining the approximation yields larger systems
The Approximating System

Using all the test elements $\phi_i \in V^h$ corresponding to “interior nodes” in $T_h$ we have:

$$Ac = b$$

where

$$a_{ij} = \int_{\Omega} \nabla \phi_j(x) \cdot \nabla \phi_i(x) \, d\Omega$$

$$b_i = \int_{\Omega} f \phi_i(x) \, d\Omega$$

Due to the local support of the basis functions $a_{ij} = 0$ unless there is a triangle that has both nodes $i$ and $j$.

- Systems are sparse
- Refining the approximation yields larger systems
Outline

1. Partial Differential Equations
   - Introduction
   - Some Example PDEs

2. How can we go about approximating PDEs?
   - Example: Poisson Problem
   - Discrete Formulation

3. About FreeFem++
   - FreeFem++ Description
   - General Program Structure

4. Sample FreeFem++ Programs
   - Poisson Problem
   - Stokes Problem

5. Advanced Topics

6. Concluding Remarks

Finite Element Approximation of Partial Differential Equations Using FreeFem++
What is FreeFem++?

- A free, open-source software package for 2-D finite element computations

- Authors: F. Hecht, O. Pironneau, A. Le Hyaric (Université Pierre et Marie Curie, Laboratoire Jacques-Louis Lions)

- Platforms: Linux, Windows, MacOS X

- Written in C++, and much of the syntax is similar to that of C++

- Includes:
  - Mesh generation and input
  - A wide range of finite elements and the ability to add new elements
  - A number of integrated linear solvers, including CG, GMRES, UMFPACK
  - Visualization tools
What is FreeFem++?

- A free, open-source software package for 2-D finite element computations
- **Authors:** F. Hecht, O. Pironneau, A. Le Hyaric (Université Pierre et Marie Curie, Laboratoire Jacques-Louis Lions)
- **Platforms:** Linux, Windows, MacOS X
- Written in C++, and much of the syntax is similar to that of C++
- **Includes:**
  - Mesh generation and input
  - A wide range of finite elements and the ability to add new elements
  - A number of integrated linear solvers, including CG, GMRES, UMFPACK
  - Visualization tools

Chrispell and Howell

Finite Element Approximation of Partial Differential Equations Using FreeFem++
What is FreeFem++?

- A free, open-source software package for 2-D finite element computations
- Authors: F. Hecht, O. Pironneau, A. Le Hyaric (Université Pierre et Marie Curie, Laboratoire Jacques-Louis Lions)
- Platforms: Linux, Windows, MacOS X
- Written in C++, and much of the syntax is similar to that of C++
- Includes:
  - Mesh generation and input
  - A wide range of finite elements and the ability to add new elements
  - A number of integrated linear solvers, including CG, GMRES, UMFPACK
  - Visualization tools
What is FreeFem++?

- A free, open-source software package for 2-D finite element computations
- Authors: F. Hecht, O. Pironneau, A. Le Hyaric (Université Pierre et Marie Curie, Laboratoire Jacques-Louis Lions)
- Platforms: Linux, Windows, MacOS X
- Written in C++, and much of the syntax is similar to that of C++
- Includes:
  - Mesh generation and input
  - A wide range of finite elements and the ability to add new elements
  - A number of integrated linear solvers, including CG, GMRES, UMFPACK
  - Visualization tools
What is FreeFem++?

- A free, open-source software package for 2-D finite element computations
- Authors: F. Hecht, O. Pironneau, A. Le Hyaric (Université Pierre et Marie Curie, Laboratoire Jacques-Louis Lions)
- Platforms: Linux, Windows, MacOS X
- Written in C++, and much of the syntax is similar to that of C++
- Includes:
  - Mesh generation and input
  - A wide range of finite elements and the ability to add new elements
  - A number of integrated linear solvers, including CG, GMRES, UMFPACK
  - Visualization tools
What is FreeFem++?

- A free, open-source software package for 2-D finite element computations
- Authors: F. Hecht, O. Pironneau, A. Le Hyaric (Université Pierre et Marie Curie, Laboratoire Jacques-Louis Lions)
- Platforms: Linux, Windows, MacOS X
- Written in C++, and much of the syntax is similar to that of C++
- Includes:
  - Mesh generation and input
  - A wide range of finite elements and the ability to add new elements
  - A number of integrated linear solvers, including CG, GMRES, UMFPACK
  - Visualization tools
What is FreeFem++?

- A free, open-source software package for 2-D finite element computations
- Authors: F. Hecht, O. Pironneau, A. Le Hyaric (Université Pierre et Marie Curie, Laboratoire Jacques-Louis Lions)
- Platforms: Linux, Windows, MacOS X
- Written in C++, and much of the syntax is similar to that of C++
- Includes:
  - Mesh generation and input
  - A wide range of finite elements and the ability to add new elements
  - A number of integrated linear solvers, including CG, GMRES, UMFPACK
  - Visualization tools
What is FreeFem++?

- A free, open-source software package for 2-D finite element computations
- Authors: F. Hecht, O. Pironneau, A. Le Hyaric (Université Pierre et Marie Curie, Laboratoire Jacques-Louis Lions)
- Platforms: Linux, Windows, MacOS X
- Written in C++, and much of the syntax is similar to that of C++
- Includes:
  - Mesh generation and input
  - A wide range of finite elements and the ability to add new elements
  - A number of integrated linear solvers, including CG, GMRES, UMFPACK
  - Visualization tools
What is FreeFem++?

- A free, open-source software package for 2-D finite element computations

- Authors: F. Hecht, O. Pironneau, A. Le Hyaric (Université Pierre et Marie Curie, Laboratoire Jacques-Louis Lions)

- Platforms: Linux, Windows, MacOS X

- Written in C++, and much of the syntax is similar to that of C++

- Includes:
  - Mesh generation and input
  - A wide range of finite elements and the ability to add new elements
  - A number of integrated linear solvers, including CG, GMRES, UMFPACK
  - Visualization tools
Why use FreeFem++?

- It's free!

- Easy to install and use

- Eliminates complicated overhead involved in programming the FEM (geometry, assembly, elements, interpolation, quadrature, etc.)

- Problems can be coded directly as variational forms

- Decent documentation and lots of examples

- Allows the user to easily test new ideas and algorithms without having to write tons of code!
Why use FreeFem++?

- It's free!
- Easy to install and use
  - Eliminates complicated overhead involved in programming the FEM (geometry, assembly, elements, interpolation, quadrature, etc.)
  - Problems can be coded directly as variational forms
  - Decent documentation and lots of examples
- Allows the user to easily test new ideas and algorithms without having to write tons of code!
Why use FreeFem++?

- It's free!
- Easy to install and use
- Eliminates complicated overhead involved in programming the FEM (geometry, assembly, elements, interpolation, quadrature, etc.)
- Problems can be coded directly as variational forms
- Decent documentation and lots of examples
- Allows the user to easily test new ideas and algorithms without having to write tons of code!
Why use FreeFem++?

- It's free!
- Easy to install and use
- Eliminates complicated overhead involved in programming the FEM (geometry, assembly, elements, interpolation, quadrature, etc.)
- Problems can be coded directly as variational forms
- Decent documentation and lots of examples
- Allows the user to easily test new ideas and algorithms without having to write tons of code!
Why use FreeFem++?

- It's free!
- Easy to install and use
- Eliminates complicated overhead involved in programming the FEM (geometry, assembly, elements, interpolation, quadrature, etc.)
- Problems can be coded directly as variational forms
- Decent documentation and lots of examples
- Allows the user to easily test new ideas and algorithms without having to write tons of code!
Why use FreeFem++?

- It's free!
- Easy to install and use
- Eliminates complicated overhead involved in programming the FEM (geometry, assembly, elements, interpolation, quadrature, etc.)
- Problems can be coded directly as variational forms
- Decent documentation and lots of examples
- Allows the user to easily test new ideas and algorithms without having to write tons of code!
Outline

1 Partial Differential Equations
   - Introduction
   - Some Example PDEs

2 How can we go about approximating PDEs?
   - Example: Poisson Problem
   - Discrete Formulation

3 About FreeFem++
   - FreeFem++ Description
   - General Program Structure

4 Sample FreeFem++ Programs
   - Poisson Problem
   - Stokes Problem

5 Advanced Topics

6 Concluding Remarks
The structure of a simple FreeFem++ program

1. Build a mesh

2. Declare the finite element space and test and trial functions from that space

3. Write the variational forms/inner products involved in the problem and construct the problem statement

4. Solve the problem

5. Analyze results (plots, error calculations, etc.)
The structure of a simple FreeFem++ program

1. Build a mesh
2. Declare the finite element space and test and trial functions from that space
3. Write the variational forms/inner products involved in the problem and construct the problem statement
4. Solve the problem
5. Analyze results (plots, error calculations, etc.)
The structure of a simple FreeFem++ program

1. Build a mesh

2. Declare the finite element space and test and trial functions from that space

3. Write the variational forms/inner products involved in the problem and construct the problem statement

4. Solve the problem

5. Analyze results (plots, error calculations, etc.)
The structure of a simple FreeFem++ program

1. Build a mesh

2. Declare the finite element space and test and trial functions from that space

3. Write the variational forms/inner products involved in the problem and construct the problem statement

4. Solve the problem

5. Analyze results (plots, error calculations, etc.)
The structure of a simple FreeFem++ program

1. Build a mesh
2. Declare the finite element space and test and trial functions from that space
3. Write the variational forms/inner products involved in the problem and construct the problem statement
4. Solve the problem
5. Analyze results (plots, error calculations, etc.)
Outline

1. Partial Differential Equations
   - Introduction
   - Some Example PDEs

2. How can we go about approximating PDEs?
   - Example: Poisson Problem
   - Discrete Formulation

3. About FreeFem++
   - FreeFem++ Description
   - General Program Structure

4. Sample FreeFem++ Programs
   - Poisson Problem
   - Stokes Problem

5. Advanced Topics

6. Concluding Remarks
FreeFem++ for the Poisson Problem

Recall the variational problem and its finite element approximation: Find $u^h \in V^h$ such that

$$a(u^h, v^h) = \int_{\Omega} (\nabla u^h) \cdot (\nabla v^h) \, d\Omega = \int_{\Omega} f \cdot v^h \, d\Omega = (f, v^h) \quad \forall \, v^h \in V^h$$

Let $\Omega = [0, 1] \times [0, 1]$ and $f$ is chosen such that

$$u(x, y) = \sin(5\pi x(1 - x)) \sin(4\pi y(1 - y))$$
FreeFem++ for the Poisson Problem

Recall the variational problem and its finite element approximation: Find $u^h \in V^h$ such that

$$a(u^h, v^h) = \int_\Omega (\nabla u^h) \cdot (\nabla v^h) \, d\Omega = \int_\Omega f \cdot v^h \, d\Omega = (f, v^h) \quad \forall v^h \in V^h$$

Let $\Omega = [0, 1] \times [0, 1]$ and $f$ is chosen such that

$$u(x, y) = \sin(5\pi x(1 - x)) \sin(4\pi y(1 - y))$$
FreeFem++ for the Poisson Problem

Recall the variational problem and its finite element approximation: *Find* $u^h \in V^h$ *such that*

$$a(u^h, v^h) = \int_{\Omega} (\nabla u^h) \cdot (\nabla v^h) \, d\Omega = \int_{\Omega} f \cdot v^h \, d\Omega = (f, v^h) \quad \forall v^h \in V^h$$

Let $\Omega = [0, 1] \times [0, 1]$ and $f$ is chosen such that

$$u(x, y) = \sin(5\pi x(1 - x)) \sin(4\pi y(1 - y))$$
Build the mesh

To build a square mesh on $\Omega = [0, 1] \times [0, 1]$, we can simply use:

```cpp
int n=10;
mesh Th=square(n,n);
```

or, more flexible code can be written:

```cpp
int n=10, m=10;
real x0=0.0, x1=1.0;
real y0=0.0, y1=0.0;
mesh Th=square(n,m,
[x0+(x1-x0)*x,y0+(y1-y0)*y]);
```
Build the mesh

To build a square mesh on \( \Omega = [0, 1] \times [0, 1] \), we can simply use:

```cpp
int n=10;
mesh Th=square(n,n);
```

or, more flexible code can be written:

```cpp
int n=10, m=10;
real x0=0.0, x1=1.0;
real y0=0.0, y1=0.0;
mesh Th=square(n,m,
            [x0+(x1-x0)*x,y0+(y1-y0)*y]);
```
Build the mesh

To build a square mesh on \( \Omega = [0, 1] \times [0, 1] \), we can simply use:

```cpp
int n=10;
mesh Th=square(n,n);
```

or, more flexible code can be written:

```cpp
int n=10, m=10;
real x0=0.0, x1=1.0;
real y0=0.0, y1=0.0;
mesh Th=square(n,m,
[x0+(x1-x0)*x,y0+(y1-y0)*y]);
```
Build the mesh

To build a square mesh on $\Omega = [0, 1] \times [0, 1]$, we can simply use:

```cpp
int n=10;
mesh Th=square(n,n);
```

or, more flexible code can be written:

```cpp
int n=10, m=10;
real x0=0.0, x1=1.0;
real y0=0.0, y1=0.0;
mesh Th=square(n,m,
    [x0+(x1-x0)*x,y0+(y1-y0)*y]);
```
Declare the FE Space and Functions

We will use $P^1$ elements for $u^h$ and $v^h$:

```markdown
fespace Vh(Th,P1);
```

Declaring functions in $V^h$ is easy:

```markdown
Vh uh, vh;
```

We specify the right-hand side and boundary functions:

```markdown
func f=-1.0*(-sin(5*pi*x*(1-x))*pow(5*pi*(1-x)-5*pi*x,2)*sin(4*pi*y*(1-y))
    -10*cos(5*pi*x*(1-x))*pi*sin(4*pi*y*(1-y))
    -sin(5*pi*x*(1-x))*sin(4*pi*y*(1-y))*pow(4*pi*(1-y)-4*pi*y,2)
    -8*sin(5*pi*x*(1-x))*cos(4*pi*y*(1-y))*pi);
```

```markdown
func g=0;
```
Declare the FE Space and Functions

We will use $P^1$ elements for $u^h$ and $v^h$:

```cpp
fespace Vh(Th,P1);
```

Declaring functions in $V^h$ is easy:

```cpp
Vh uh, vh;
```

We specify the right-hand side and boundary functions:

```cpp
func f=-1.0*(-sin(5*pi*x*(1-x))*pow(5*pi*(1-x)-5*pi*x,2)*sin(4*pi*y*(1-y))
    -10*cos(5*pi*x*(1-x))*pi*sin(4*pi*y*(1-y))
    -sin(5*pi*x*(1-x))*sin(4*pi*y*(1-y))*pow(4*pi*(1-y)-4*pi*y,2)
    -8*sin(5*pi*x*(1-x))*cos(4*pi*y*(1-y))*pi);
func g=0;
```
Declare the FE Space and Functions

We will use \( P^1 \) elements for \( u^h \) and \( v^h \):

\[
\text{fespace } Vh(Th,P1);
\]

Declaring functions in \( V^h \) is easy:

\[
Vh \ uh, \ vh;
\]

We specify the right-hand side and boundary functions:

\[
\begin{align*}
\text{func } f &= -1.0*(-\sin(5\pi x(1-x))*\text{pow}(5\pi x(1-x)-5\pi x,2)*\sin(4\pi y(1-y)) \\
&\quad -10*\cos(5\pi x(1-x))*\pi*\sin(4\pi y(1-y)) \\
&\quad -\sin(5\pi x(1-x))*\sin(4\pi y(1-y))*\text{pow}(4\pi *(1-y)-4\pi y,2) \\
&\quad -8*\sin(5\pi x(1-x))*\cos(4\pi y(1-y))*\pi);
\end{align*}
\]

\[
\text{func } g = 0;
\]
Declare the FE Space and Functions

We will use $P^1$ elements for $u^h$ and $v^h$:

```cpp
fespace Vh(Th,P1);
```

Declaring functions in $V^h$ is easy:

```cpp
Vh uh, vh;
```

We specify the right-hand side and boundary functions:

```cpp
func f=-1.0*(-sin(5*pi*x*(1-x))*pow(5*pi*(1-x)-5*pi*x,2)*sin(4*pi*y*(1-y))
    -10*cos(5*pi*x*(1-x))*pi*sin(4*pi*y*(1-y))
    -sin(5*pi*x*(1-x))*sin(4*pi*y*(1-y))*pow(4*pi*(1-y)-4*pi*y,2)
    -8*sin(5*pi*x*(1-x))*cos(4*pi*y*(1-y))*pi);
func g=0;
```
Write the Variational Forms and Problem Statement

We can code

\[ a(u^h, v^h) = \int_\Omega (\nabla u^h) \cdot (\nabla v^h) \, d\Omega = \int_\Omega (u_x v_x + u_y v_y) \, d\Omega \]

with

\[ \text{int2d}(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh)) \]

and \((f, v^h)\) as

\[ \text{int2d}(Th)(f*vh) \]

Then by adding the boundary conditions, we can write our problem statement:

\[ \text{problem poisson(uh,vh) = int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))} \]
\[ -\text{int2d}(Th)(f*vh) \]
\[ + \text{on(1,2,3,4,uh=g)} ; \]
Write the Variational Forms and Problem Statement

We can code

\[ a(u^h, v^h) = \int_{\Omega} (\nabla u^h) \cdot (\nabla v^h) \, d\Omega = \int_{\Omega} (u_x v_x + u_y v_y) \, d\Omega \]

with

\[
\text{int2d}(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))
\]

and \((f, v^h)\) as

\[
\text{int2d}(Th)(f*vh)
\]

Then by adding the boundary conditions, we can write our problem statement:

\[
\text{problem poisson(uh,vh) = int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh)) -int2d(Th)(f*vh) + on(1,2,3,4,uh=g) ;}
\]
Write the Variational Forms and Problem Statement

We can code

\[ a(u^h, v^h) = \int_{\Omega} (\nabla u^h) \cdot (\nabla v^h) \, d\Omega = \int_{\Omega} (u_x v_x + u_y v_y) \, d\Omega \]

with

\[ \text{int2d}(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh)) \]

and \((f, v^h)\) as

\[ \text{int2d}(Th)(f*vh) \]

Then by adding the boundary conditions, we can write our problem statement:

\[ \text{problem poisson(uh,vh) = int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh)) -int2d(Th)(f*vh) + on(1,2,3,4,uh=g) ;} \]
Solving the Problem and Viewing the Solution

The problem is solved by simply executing the problem statement:

```plaintext
poisson;
```

Then we can plot the solution:

```plaintext
plot(uh,fill=1,value=1);
```
Solving the Problem and Viewing the Solution

The problem is solved by simply executing the problem statement:

```
poisson;
```

Then we can plot the solution:

```
plot(uh,fill=1,value=1);
```
The problem is solved by simply executing the problem statement:

```cpp
poisson;
```

Then we can plot the solution:

```cpp
plot(uh, fill=1, value=1);
```
Error Calculations

As this problem has an analytic solution, we can compute the $L^2$ and $H^1$ errors associated with our approximation. First define the true solution and its derivatives:

```cpp
func utrue=sin(5*pi*x*(1-x))*sin(4*pi*y*(1-y));
func utruex=cos(5*pi*x*(1-x))*(5*pi*(1-x)-5*pi*x)*sin(4*pi*y*(1-y));
func utruey=sin(5*pi*x*(1-x))*cos(4*pi*y*(1-y))*(4*pi*(1-y)-4*pi*y);
```

Then we can compute the quantities

$$\|u - u^h\|_0 \quad \text{and} \quad \|u - u^h\|_1$$

and print the errors:

```cpp
real ul2 = sqrt(int2d(Th)((utrue-uh)^2));
real uh1 = sqrt(int2d(Th)(ul2^2 + (utruey-dx(uh))^2+(utruey-dy(uh))^2));
cout << "u L^2 error: " << ul2 << endl;
cout << "u H^1 error: " << uh1 << endl;
```
Error Calculations

As this problem has an analytic solution, we can compute the $L^2$ and $H^1$ errors associated with our approximation. First define the true solution and its derivatives:

```cpp
func utrue=sin(5*pi*x*(1-x))*sin(4*pi*y*(1-y));
func utruex=cos(5*pi*x*(1-x))*(5*pi*(1-x)-5*pi*x)*sin(4*pi*y*(1-y));
func utruvey=sin(5*pi*x*(1-x))*cos(4*pi*y*(1-y))*(4*pi*(1-y)-4*pi*y);
```

Then we can compute the quantities

$$\|u - u^h\|_0 \quad \text{and} \quad \|u - u^h\|_1$$

and print the errors:

```cpp
real ul2 = sqrt(int2d(Th)((utrue-uh)^2));
real uh1 = sqrt(int2d(Th)(ul2^2 + (utruex-dx(uh))^2+(utruey-dy(uh))^2));

cout << "u L^2 error: " << ul2 << endl;
cout << "u H^1 error: " << uh1 << endl;
```
Error Calculations

As this problem has an analytic solution, we can compute the $L^2$ and $H^1$ errors associated with our approximation. First define the true solution and its derivatives:

```cpp
func utrue=sin(5*pi*x*(1-x))*sin(4*pi*y*(1-y));
func utruex=cos(5*pi*x*(1-x))*(5*pi*(1-x)-5*pi*x)*sin(4*pi*y*(1-y));
func utruey=sin(5*pi*x*(1-x))*cos(4*pi*y*(1-y))*(4*pi*(1-y)-4*pi*y);
```

Then we can compute the quantities

$$\|u - u^h\|_0 \quad \text{and} \quad \|u - u^h\|_1$$

and print the errors:

```cpp
real ul2 = sqrt(int2d(Th)((utrue-uh)^2));
real uh1 = sqrt(int2d(Th)(ul2^2 + (utruex-dx(uh))^2 + (utruey-dy(uh))^2));
cout << "u L^2 error: " << ul2 << endl;
cout << "u H^1 error: " << uh1 << endl;
```
Error Calculations

As this problem has an analytic solution, we can compute the $L^2$ and $H^1$ errors associated with our approximation. First define the true solution and its derivatives:

```cpp
func utrue=sin(5*pi*x*(1-x))*sin(4*pi*y*(1-y));
func utruex=cos(5*pi*x*(1-x))*(5*pi*(1-x)-5*pi*x)*sin(4*pi*y*(1-y));
func utruey=sin(5*pi*x*(1-x))*cos(4*pi*y*(1-y))*(4*pi*(1-y)-4*pi*y);
```

Then we can compute the quantities

$$\|u - u^h\|_0 \quad \text{and} \quad \|u - u^h\|_1$$

and print the errors:

```cpp
real ul2 = sqrt(int2d(Th)((utrue-uh)^2));
real uh1 = sqrt(int2d(Th)(ul2^2 + (utruex-dx(uh))^2+(utruey-dy(uh))^2));
cout << "u L^2 error: " << ul2 << endl;
cout << "u H^1 error: " << uh1 << endl;
```
The Entire Program

```plaintext
int n=10;
mesh Th=square(n,n);
fespace Vh(Th,P1);
Vh uh, vh;
func f=-1.0*(-sin(5*pi*x*(1-x))*pow(5*pi*(1-x)-5*pi*x,2)*sin(4*pi*y*(1-y))
    -10*cos(5*pi*x*(1-x))*pi*sin(4*pi*y*(1-y))
    -sin(5*pi*x*(1-x))*sin(4*pi*y*(1-y))*pow(4*pi*(1-y)-4*pi*y,2)
    -8*sin(5*pi*x*(1-x))*cos(4*pi*y*(1-y))*pi);
func g=0;
problem poisson(uh,vh) = int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))
    -int2d(Th)(f*vh)
    + on(1,2,3,4,uh=g) ;
poisson;
plot(uh,fill=1,value=1);
func utrue=sin(5*pi*x*(1-x))*sin(4*pi*y*(1-y));
func utruex=cos(5*pi*x*(1-x))*(5*pi*(1-x)-5*pi*x)*sin(4*pi*y*(1-y));
func utruey=sin(5*pi*x*(1-x))*cos(4*pi*y*(1-y))*(4*pi*(1-y)-4*pi*y);
real ul2 = sqrt(int2d(Th)((utrue-uh)^2));
real uh1 = sqrt(int2d(Th)(ul2^2 + (utruex-dx(uh))^2+(utruey-dy(uh))^2));
cout << "u L^2 error: " << ul2 << endl;
cout << "u H^1 error: " << uh1 << endl;
```

Chrispell and Howell

Finite Element Approximation of Partial Differential Equations Using FreeFem++
Convergence to the Exact Solution

As theory predicts, we have

\[ \| u - u^h \|_0 \leq C h^2 \quad \text{and} \quad \| u - u^h \|_1 \leq C h \]

<table>
<thead>
<tr>
<th>( n )</th>
<th>( | u - u^h |_0 )</th>
<th>rate</th>
<th>( | u - u^h |_1 )</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.087719</td>
<td></td>
<td>2.43762</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.024366</td>
<td>1.85</td>
<td>1.27541</td>
<td>0.93</td>
</tr>
<tr>
<td>40</td>
<td>0.006278</td>
<td>1.96</td>
<td>0.64626</td>
<td>0.98</td>
</tr>
<tr>
<td>80</td>
<td>0.001582</td>
<td>1.99</td>
<td>0.32425</td>
<td>1.00</td>
</tr>
<tr>
<td>160</td>
<td>0.000396</td>
<td>2.00</td>
<td>0.16227</td>
<td>1.00</td>
</tr>
<tr>
<td>320</td>
<td>0.000099</td>
<td>2.00</td>
<td>0.08115</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Plot of Solution on Finest Mesh

Finite Element Approximation of Partial Differential Equations Using FreeFem++
Outline

1 Partial Differential Equations
   - Introduction
   - Some Example PDEs

2 How can we go about approximating PDEs?
   - Example: Poisson Problem
   - Discrete Formulation

3 About FreeFem++
   - FreeFem++ Description
   - General Program Structure

4 Sample FreeFem++ Programs
   - Poisson Problem
   - Stokes Problem

5 Advanced Topics

6 Concluding Remarks
Description of the Stokes Problem

Recall the discrete Stokes problem: Find \((u^h, p^h)\) where

\[
\begin{align*}
    a(u^h, v^h) + b(p^h, v^h) &= (f, v^h) \quad \forall v^h \in V^h, \\
    b(q^h, u^h) &= 0 \quad \forall q^h \in Q^h.
\end{align*}
\]

Here

\[
a(u^h, v^h) = \int_{\Omega} \nabla u^h : \nabla v^h \, d\Omega = \int_{\Omega} (u_{1,x}^h v_{1,x}^h + u_{2,x}^h v_{2,x}^h + u_{1,y}^h v_{1,y}^h + u_{2,y}^h v_{2,y}^h) \, d\Omega
\]

and

\[
b(q^h, u^h) = \int_{\Omega} q^h \text{div} u^h \, d\Omega = \int_{\Omega} q^h (u_{1,x}^h + u_{2,y}^h) \, d\Omega
\]
Description of the Stokes Problem

Recall the discrete Stokes problem: Find \((u^h, p^h)\) where

\[
\begin{align*}
 a(u^h, v^h) + b(p^h, v^h) &= (f, v^h) \quad \forall v^h \in \mathbf{V}^h, \\
 b(q^h, u^h) &= 0 \quad \forall q^h \in \mathbf{Q}^h.
\end{align*}
\]

Here

\[
\begin{align*}
 a(u^h, v^h) &= \int_{\Omega} \nabla u^h : \nabla v^h \, d\Omega = \int_{\Omega} \left( u_{1,x}^h v_{1,x}^h + u_{2,x}^h v_{2,x}^h + u_{1,y}^h v_{1,y}^h + u_{2,y}^h v_{2,y}^h \right) \, d\Omega \\
 b(q^h, u^h) &= \int_{\Omega} q^h \, \text{div} \, u^h \, d\Omega = \int_{\Omega} q^h \left( u_{1,x}^h + u_{2,y}^h \right) \, d\Omega
\end{align*}
\]
The Driven Cavity

- A square cavity is filled with fluid.
- The horizontal velocity at the top of the cavity is set to 1.
- \( f = 0 \)

\( \mathbf{u}_{\Gamma_{\text{top}}} = [1, 0]^T \)

\( \mathbf{u}_{\Gamma \setminus \Gamma_{\text{top}}} = [0, 0]^T \)
The Driven Cavity

- A square cavity is filled with fluid

- The horizontal velocity at the top of the cavity is set to 1

- \( f = 0 \)
The Driven Cavity

- A square cavity is filled with fluid.

- The horizontal velocity at the top of the cavity is set to 1.

- \( f = 0 \)
The Driven Cavity

- A square cavity is filled with fluid

- The horizontal velocity at the top of the cavity is set to 1

- $f = 0$
The Driven Cavity

- A square cavity is filled with fluid
- The horizontal velocity at the top of the cavity is set to 1
- \( f = 0 \)
The Driven Cavity

- A square cavity is filled with fluid
- The horizontal velocity at the top of the cavity is set to 1
- $f = 0$

$$\mathbf{u}_{\Gamma_{\text{top}}} = [1, 0]^T$$

$$\mathbf{u}_{\Gamma \setminus \Gamma_{\text{top}}} = [0, 0]^T$$
FreeFem++ code for Stokes Driven Cavity Problem

```c
int n=3;
mesh Th=square(10*n,10*n);

fespace Vh(Th,P1b);
fespace Qh(Th,P1);

Vh u1,u2,v1,v2;
Qh p,q;

solve stokes([[u1,u2,p]],[[v1,v2,q]]) =
  int2d(Th)(dx(u1)*dx(v1)+dy(u1)*dy(v1)
  + dx(u2)*dx(v2)+ dy(u2)*dy(v2)
  + dx(p)*v1 + dy(p)*v2 + q*(dx(u1)+dy(u2)))
  + on(1,2,4,u1=0,u2=0) + on(3,u1=1,u2=0);

plot(p,[u1,u2],fill=1);
```

Chrispell and Howell
Finite Element Approximation of Partial Differential Equations Using FreeFem++
Driven Cavity Problem Results
Mixed Finite Elements used for Stokes

The spaces $V^h$ and $Q^h$ have to be chosen so that they satisfy the inf-sup condition:

$$\inf_{0 \neq q^h \in Q^h} \sup_{0 \neq v^h \in V^h} \frac{(q^h, \nabla \cdot v^h)}{\|v^h\|_1 \|q^h\|_0} \geq C$$

One choice of $V^h$ and $Q^h$ that satisfies the condition is:

```cpp
fespace Vh(Th,P1b);
fespace Qh(Th,P1);
```

i.e., $V^h = \{ v \in V : v|_K = (P^1_b(K))^2 \}$ and $Q^h = \{ q \in Q : q|_K = P^1(K) \}$.

Warning: using elements that do not satisfy the mathematical framework can produce disastrous results!
Mixed Finite Elements used for Stokes

The spaces $V^h$ and $Q^h$ have to be chosen so that they satisfy the inf-sup condition:

$$\inf_{0 \neq q^h \in Q^h} \sup_{0 \neq v^h \in V^h} \frac{(q^h, \nabla \cdot v^h)}{\|v^h\|_1 \|q^h\|_0} \geq C$$

One choice of $V^h$ and $Q^h$ that satisfies the condition is:

```plaintext
fespace Vh(Th,P1b);
fespace Qh(Th,P1);
```

i.e., $V^h = \{ v \in V : v|_K = (P^1_b(K))^2 \}$ and $Q^h = \{ q \in Q : q|_K = P^1(K) \}$.

Warning: using elements that do not satisfy the mathematical framework can produce disastrous results!
Mixed Finite Elements used for Stokes

The spaces $V^h$ and $Q^h$ have to be chosen so that they satisfy the inf-sup condition:

$$\inf_{0 \neq q^h \in Q^h} \sup_{0 \neq v^h \in V^h} \frac{(q^h, \nabla \cdot v^h)}{\|v^h\|_1 \|q^h\|_0} \geq C$$

One choice of $V^h$ and $Q^h$ that satisfies the condition is:

```plaintext
fespace Vh(Th,P1b);
fespace Qh(Th,P1);
```

i.e., $V^h = \{ v \in V : v|_K = (P^1_b(K))^2 \}$ and $Q^h = \{ q \in Q : q|_K = P^1(K) \}$.

Warning: using elements that do not satisfy the mathematical framework can produce disastrous results!
An Example of Incompatible Elements

One choice of $V^h$ and $Q^h$ that does NOT satisfy the compatibility condition is:

```c
fespace Vh(Th,P1);
fespace Qh(Th,P1);
```
An Example of Incompatible Elements

One choice of $\mathbf{V}^h$ and $Q^h$ that does NOT satisfy the compatibility condition is:

```plaintext
fespace Vh(Th,P1);
fespace Qh(Th,P1);
```

Chrispell and Howell

Finite Element Approximation of Partial Differential Equations Using FreeFem++
Some Advanced Topics

- Nonlinear Problems
- Input/Output
- Adaptive Mesh Refinement
- Scripting/GNUplot
- Advanced Meshing
- Skip Advanced Topics
Nonlinear Problems

- For nonlinear PDEs, the discrete problem results in a nonlinear system of equations
- To solve this system of nonlinear equations, an iterative method is required, such as Newton’s Method
- You can use the Fréchet Derivative to linearize a nonlinear system about a known solution
- Then construct a linear approximation to the original problem using the derivative
**Nonlinear Problems**

- For nonlinear PDEs, the discrete problem results in a nonlinear system of equations.

- To solve this system of nonlinear equations, an iterative method is required, such as Newton’s Method.

- You can use the Fréchet Derivative to linearize a nonlinear system about a known solution.

- Then construct a linear approximation to the original problem using the derivative.
For nonlinear PDEs, the discrete problem results in a nonlinear system of equations.

To solve this system of nonlinear equations, an iterative method is required, such as Newton’s Method.

You can use the Fréchet Derivative to linearize a nonlinear system about a known solution.

Then construct a linear approximation to the original problem using the derivative.
Nonlinear Problems

- For nonlinear PDEs, the discrete problem results in a nonlinear system of equations.

- To solve this system of nonlinear equations, an iterative method is required, such as Newton’s Method.

- You can use the Fréchet Derivative to linearize a nonlinear system about a known solution.

- Then construct a linear approximation to the original problem using the derivative.
The Steady Navier-Stokes Equations

The original steady-state Navier-Stokes discrete problem: Find \((u^h, p^h)\) where

\[
a(u^h, v^h) + b(p^h, v^h) + (u^h \cdot \nabla u^h, v^h) + b(q^h, u^h) = (f, v^h) \quad \forall (v^h, q^h) \in \mathbf{V}^h \times \mathbf{Q}^h.
\]

Linearization of the problem for use in a Newton Iteration: Given \((u^h_0, p^h_0)\), for \(i = 1, 2, \ldots\), find \((u^h_i, p^h_i)\) where

\[
a(u^h_i, v^h) + b(p^h_i, v^h) + (u^h_i \cdot \nabla u^h_{i-1}, v^h) + (u^h_i \cdot \nabla u^h_{i-1}, v^h) + b(q^h_i, u^h_i) = (f, v^h) + (u^h_{i-1} \cdot \nabla u^h_{i-1}, v^h) \quad \forall (v^h, q^h) \in \mathbf{V}^h \times \mathbf{Q}^h.
\]
The Steady Navier-Stokes Equations

The original steady-state Navier-Stokes discrete problem: Find \((u^h, p^h)\) where

\[
a(u^h, v^h) + b(p^h, v^h) + (u^h \cdot \nabla u^h, v^h) + b(q^h, u^h)
= (f, v^h) \quad \forall (v^h, q^h) \in V^h \times Q^h.
\]

Linearization of the problem for use in a Newton Iteration: Given \((u_0^h, p_0^h)\), for \(i = 1, 2, \ldots\), find \((u_i^h, p_i^h)\) where

\[
a(u_i^h, v^h) + b(p_i^h, v^h) + (u_i^h \cdot \nabla u_{i-1}^h, v^h) + (u_i^h \cdot \nabla u_{i-1}^h, v^h) + b(q^h, u_i^h)
= (f, v^h) + (u_{i-1}^h \cdot \nabla u_{i-1}^h, v^h) \quad \forall (v^h, q^h) \in V^h \times Q^h.
\]
FreeFem++ code for Navier-Stokes Nonlinear Iteration

```plaintext
Vh u1,u2,v1,v2,u1o,u2o;
Qh p,q;

problem navierstokes([u1,u2,p],[v1,v2,q]) =
  int2d(Th)( dx(u1)*dx(v1)+dy(u1)*dy(v1)+ dx(u2)*dx(v2)+ dy(u2)*dy(v2)
  + dx(p)*v1 + dy(p)*v2 + q*(dx(u1)+dy(u2))
  + (u1*dx(u1o)+u2*dy(u1o))*v1 + (u1*dx(u2o)+u2*dy(u2o))*v2
  + (u1o*dx(u1)+u2o*dy(u1))*v1 + (u1o*dx(u2)+u2o*dy(u2))*v2
  - (u1o*dx(u1o)+u2o*dy(u1o))*v1 - (u1o*dx(u2o)+u2o*dy(u2o))*v2
  - (f1*v1 + f2*v2)
  + on(1,2,3,4,u1=0,u2=0);

u1 = 0;
u2 = 0;

for(i=0;i<=10;i++) {
  u1o = u1;
u2o = u2;
  navierstokes;
}
```
Sample file manipulation:

```cpp
ofstream uout("./data/u2.6.out");
uout << u[];
ifstream uin("./data/u2.6.out");
uin >> u[];
```

Sample command-line input and output:

```cpp
int i, j, n, m;
real d=2.0, xx=0.0, yy=0.0;
cout << "Enter the number of x and y data points desired: " << endl;
cin >> n >> m;
func f=sin(d*pi*x)*cos((d+1)*pi*y);
for (j=0;j<m;j++) {
    yy = 1.0*j/(m-1);
    for (i=0;i<n;i++) {
        xx = 1.0*i/(n-1);
        cout << f(xx,yy) << " ";
    }
    cout << endl;
}
```

Finite Element Approximation of Partial Differential Equations Using FreeFem++
Input/Output

Sample file manipulation:

```cpp
ofstream uout("./data/u2.6.out");
uout << u[];
ifstream uin("./data/u2.6.out");
uin >> u[];
```

Sample command-line input and output:

```cpp
int i, j, n, m;
real d=2.0, xx=0.0, yy=0.0;
cout << "Enter the number of x and y data points desired: " << endl;
cin >> n >> m;
func f=sin(d*pi*x)*cos((d+1)*pi*y);
for (j=0;j<m;j++) {
  yy = 1.0*j/(m-1);
  for (i=0;i<n;i++) {
    xx = 1.0*i/(n-1);
    cout << f(xx,yy) << " ";
  }
  cout << endl;
}
```
FreeFem++ has built-in mesh adaptivity routines.

```plaintext
func f = 10.0*x^3+y^3 +atan2(0.0001,sin(5.0*y)-2.0*x);
mesh Th=square(5,5,[-1+2*x,-1+2*y]);
fespace Vh(Th,P1);
Vh fh=f;
plot(fh);
for (int i=0;i<2;i++) {
    Th=adaptmesh(Th,fh);
    fh=f;
    plot(Th,fh);
}
```
FreeFem++ has built-in mesh adaptivity routines.

```cpp
func f = 10.0*x^3+y^3 +atan2(0.0001,sin(5.0*y)-2.0*x);
mesh Th=square(5,5,[-1+2*x,-1+2*y]);
fespace Vh(Th,P1);
Vh fh=f;
plot(fh);
for (int i=0;i<2;i++) {
    Th=adaptmesh(Th,fh);
    fh=f;
    plot(Th,fh);
}
```
FreeFem++ will allow for command line scripting:

```cpp
string plotdata = "poisson" + n + ".sol";
{ofstream PlotFile(plotdata);
  for (int i=0; i <=n ; i++){
    for (int j=0; j<=n ; j++){
      PlotFile << (0.0+i*(1.0/n))
        << " " << (0.0+j*(1.0/n))
        << " " << uh( (0.0+i*(1.0/n)), (0.0+j*(1.0/n)))
        << endl;
    }
  }
  PlotFile << endl;
}
exec("echo ' set parametric \n" +
  " set term postscript eps enhanced color solid \n" +
  " set hidden \n" +
  " set contour base \n" +
  " set data style lines \n " +
  " set output "+ plotdata + ".eps" \n" +
  " splot "+ plotdata + "" \n" +
  "'|gnuplot ");
```

Adding this code to the Poisson example produces:

Useful for:
- 3D plotting
- \LaTeX error reports
- Calling C or other code.
FreeFem++ will allow for command line scripting:

```cpp
string plotdata = "poisson" + n + ".sol";
{ofstream PlotFile(plotdata);
 for (int i=0; i < n; i++){
   for (int j=0; j < n; j++){
     PlotFile << (0.0+i*(1.0/n))
         << " " << (0.0+j*(1.0/n))
         << " " << uh( (0.0+i*(1.0/n))
             , (0.0+j*(1.0/n)))
         << endl;
   }
   PlotFile << endl;
 }
exec("echo ' set parametric \
       " set term postscript eps 
        enhanced color solid \
         " set hidden \
         " set contour base \
         " set data style lines \
         " set output '"" + plotdata + ".eps" "
        " splot '"" + plotdata + "" "
        " \|gnuplot ");
```

Adding this code to the Poisson example produces:

Useful for:
- 3D plotting
- \textit{\LaTeX} error reports
- Calling C or other code.
FreeFem++ will allow for command line scripting:

```cpp
string plotdata = "poisson" + n + ".sol";
{ofstream PlotFile(plotdata);
  for (int i=0; i <=n ; i++){
    for (int j=0; j<=n ; j++){
      PlotFile << (0.0+i*(1.0/n)) << " " << (0.0+j*(1.0/n)) << " " << uh( (0.0+i*(1.0/n)), (0.0+j*(1.0/n))) << endl;
    }
  PlotFile << endl;
}
exec("echo ' set parametric \n" +
  " set term postscript eps enhanced color solid \n" +
  " set hidden \n" +
  " set contour base \n" +
  " set data style lines \n" +
  " set output " + plotdata + ".eps\n" +
  " splot " + plotdata + "\n" |
gnuplot ");
```

Adding this code to the Poisson example produces:

Useful for:
- 3D plotting
- \LaTeX\ error reports
- Calling C or other code.
Advanced Meshing

Meshes can be created by parametrizations of the boundary:

```plaintext
border a0(t=0,1){x=2*t; y=0; label=1;}
border a1(t=0,1){x=2+1.5*t; y=0; label=1;}
border a2(t=0,1){x=3.5*t; y=0; label=1;}
border a3(t=0,1){x=4.5+3.5*t; y=0; label=1;}
border a4(t=0,1){x=8; y=0.125*t; label=2;}
border a5(t=0,1){x=8; y=0.125+0.125*t;label=2;}
border a6(t=0,1){x=8-3.5*t; y=0.25; label=3;}
border a7(t=0,1){x=4.5-0.5*t; y=0.25; label=3;}
border a8(t=0,1){x=4; y=0.25+0.125*t; label=4;}
border a9(t=0,1){x=4; y=0.375+0.25*t; label=4;}
border a10(t=0,1){x=4; y=0.625+0.375*t;label=4;}
border a11(t=0,1){x=4-0.5*t; y=1; label=5;}
border a12(t=0,1){x=3.5-1.5*t; y=1; label=5;}
border a13(t=0,1){x=2-2*t; y=1; label=5;}
border a14(t=0,1){x=0; y=1-0.375*t; label=6;}
border a15(t=0,1){x=0; y=0.625-0.25*t;label=6;}
border a16(t=0,1){x=0; y=0.375-0.25*t; label=6;}
border a17(t=0,1){x=0; y=0.125-0.125*t;label=6;}
n=3;
Th= buildmesh(a0(4*n)+a1(4*n)+a2(8*n)+a3(4*n)//bottom edge
+a4(2*n)+a5(4*n)//outflow edge
+a6(4*n)+a7(4*n)//top of contraction channel
+a8(4*n)+a9(4*n)+a10(4*n)//contraction wall
+a11(4*n)+a12(4*n)+a13(4*n)//top of inflow channel
+a14(4*n)+a15(4*n)+a16(8*n)+a17(2*n));//inflow wall*/
```
Contraction Flow Mesh

The top half of a 4:1 contraction flow for fluids:
Advanced Meshing, part 2

You can even create really cool meshes:
You can even create really cool meshes:
Concluding Remarks

- The Finite Element Method provides a very nice mathematical framework for the numerical approximation of partial differential equations.

- Implementing the FEM in code requires substantial programming effort and complexity - although everyone who uses the FEM should know “how” to implement it!

- FreeFem++ provides a way to use the FEM without a substantial investment of time in programming.

- FreeFem++ is highly flexible and allows easy implementation of new algorithms or ideas in the numerical approximation of PDEs.
Concluding Remarks

- The Finite Element Method provides a very nice mathematical framework for the numerical approximation of partial differential equations.

- Implementing the FEM in code requires substantial programming effort and complexity - although everyone who uses the FEM should know “how” to implement it!

- FreeFem++ provides a way to use the FEM without a substantial investment of time in programming.

- FreeFem++ is highly flexible and allows easy implementation of new algorithms or ideas in the numerical approximation of PDEs.
Concluding Remarks

- The Finite Element Method provides a very nice mathematical framework for the numerical approximation of partial differential equations.

- Implementing the FEM in code requires substantial programming effort and complexity - although everyone who uses the FEM should know “how” to implement it!

- FreeFem++ provides a way to use the FEM without a substantial investment of time in programming.

- FreeFem++ is highly flexible and allows easy implementation of new algorithms or ideas in the numerical approximation of PDEs.
Concluding Remarks

- The Finite Element Method provides a very nice mathematical framework for the numerical approximation of partial differential equations.

- Implementing the FEM in code requires substantial programming effort and complexity - although everyone who uses the FEM should know “how” to implement it!

- **FreeFem++** provides a way to use the FEM without a substantial investment of time in programming.

- **FreeFem++** is highly flexible and allows easy implementation of new algorithms or ideas in the numerical approximation of PDEs.