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

Outline

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

Laplace's Equation

Find u such that:

$$\begin{aligned} \Delta u &= 0, \quad \mathbf{x} \text{ in } \Omega \\ u(x) &= g, \quad \mathbf{x} \text{ on } \partial\Omega \end{aligned}$$

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

Notation:

$$\begin{aligned} \Omega &\subset \mathbb{R}^d, \text{ for } d \in \{1,2,3\} \\ \partial\Omega = \Gamma &= \text{Boundary of } \Omega \\ \nabla &= \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \end{bmatrix}^T \\ \Delta &= \nabla \cdot \nabla \end{aligned}$$

Laplace's Equation

Find u such that:

$$\begin{aligned} \Delta u &= 0, \quad \mathbf{x} \text{ in } \Omega \\ u(x) &= g, \quad \mathbf{x} \text{ on } \partial\Omega \end{aligned}$$

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

Notation:

$$\begin{aligned} \Omega &\subset \mathbb{R}^d, \text{ for } d \in \{1,2,3\} \\ \partial\Omega = \Gamma &= \text{Boundary of } \Omega \\ \nabla &= \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \end{bmatrix}^T \\ \Delta &= \nabla \cdot \nabla \end{aligned}$$

Convection-Diffusion Problem

Find u such that:

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

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

Convection-Diffusion Problem

Find u such that:

$$\begin{aligned} -\Delta u + \mathbf{b} \cdot \nabla u + cu &= f, \; \mathbf{x} \in \Omega \\ u &= g, \; \mathbf{x} \text{ on } \partial\Omega. \end{aligned}$$

- 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 Ω

Convection-Diffusion Problem

Find u such that:

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

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

Stokes Problem

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

$$\begin{aligned} -\Delta \mathbf{u} + \nabla p &= \mathbf{f}, \quad \mathbf{x} \in \Omega \\ \nabla \cdot \mathbf{u} &= 0, \quad \mathbf{x} \in \Omega \\ \mathbf{u} &= g, \quad \mathbf{x} \text{ on } \partial\Omega \end{aligned}$$

- 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:

$$\begin{aligned} -\Delta \mathbf{u} + \nabla p &= \mathbf{f}, \quad \mathbf{x} \in \Omega \\ \nabla \cdot \mathbf{u} &= 0, \quad \mathbf{x} \in \Omega \\ \mathbf{u} &= g, \quad \mathbf{x} \text{ on } \partial\Omega \end{aligned}$$

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

Stokes Problem

Find **u** and p such that:

$$
\begin{aligned}
-\Delta \mathbf{u} + \nabla p &= \mathbf{f}, \quad \mathbf{x} \in \Omega \\
\nabla \cdot \mathbf{u} &= 0, \quad \mathbf{x} \in \Omega \\
\mathbf{u} &= g, \quad \mathbf{x} \text{ on } \partial\Omega
\end{aligned}
$$

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

Stokes Problem

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

$$\begin{aligned} -\Delta \mathbf{u} + \nabla p &= \mathbf{f}, \quad \mathbf{x} \in \Omega \\ \nabla \cdot \mathbf{u} &= 0, \quad \mathbf{x} \in \Omega \\ \mathbf{u} &= g, \quad \mathbf{x} \text{ on } \partial\Omega \end{aligned}$$

- 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")

Navier-Stokes Equations

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

$$\begin{aligned} Re\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) - \Delta\mathbf{u} + \nabla p &= \mathbf{f}, \quad \mathbf{x}\in\Omega \\ \nabla\cdot\mathbf{u} &= 0, \quad \mathbf{x}\in\Omega \\ \mathbf{u} &= g, \quad \mathbf{x} \text{ on } \partial\Omega \end{aligned}$$

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

$$\begin{aligned} Re\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) - \Delta\mathbf{u} + \nabla p &= \mathbf{f}, \quad \mathbf{x}\in\Omega \\ \nabla\cdot\mathbf{u} &= 0, \quad \mathbf{x}\in\Omega \\ \mathbf{u} &= g, \quad \mathbf{x} \text{ on } \partial\Omega \end{aligned}$$

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

$$\begin{aligned} Re\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) - \Delta\mathbf{u} + \nabla p &= \mathbf{f}, \quad \mathbf{x}\in\Omega \\ \nabla\cdot\mathbf{u} &= 0, \quad \mathbf{x}\in\Omega \\ \mathbf{u} &= g, \quad \mathbf{x} \text{ on } \partial\Omega \end{aligned}$$

- 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

Outline

Notation

Funciton Spaces:

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

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

$$V = H_0^1(\Omega) = \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega \right\} \quad (3)$$

Respective inner products and norms:

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

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

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

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

Notation

Funciton Spaces:

$$
\begin{aligned}
L^2(\Omega) &= \left\{ v \in \Omega : \int_\Omega v^2 \, d\Omega < \infty \right\} && (1) \\
H^1(\Omega) &= \left\{ v \in L^2(\Omega) : \nabla u \in L^2(\Omega) \right\} && (2) \\
V &= H_0^1(\Omega) = \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega \right\} && (3)
\end{aligned}
$$

Respective inner products and norms:

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

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

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

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

Notation

Funciton Spaces:

$$
\begin{aligned}
L^2(\Omega) &= \left\{ v \in \Omega : \int_\Omega v^2 \, d\Omega < \infty \right\} && (1)\\
H^1(\Omega) &= \left\{ v \in L^2(\Omega) : \nabla u \in L^2(\Omega) \right\} && (2)\\
V &= H_0^1(\Omega) = \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega \right\} && (3)
\end{aligned}
$$

Respective inner products and norms:

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

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

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

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

The Poisson Problem

Find u such that:

$$\begin{aligned} -\Delta u &= f, \quad \mathbf{x} \in \Omega \subset \mathbb{R}^2 \\ u &= 0, \quad \text{on } \partial\Omega. \end{aligned}$$

The Poisson Problem

Find u such that:

$$\begin{aligned} -\Delta u &= f, \quad \mathbf{x} \in \Omega \subset \mathbb{R}^2 \\ u &= 0, \quad \text{on } \partial\Omega. \end{aligned}$$

How do we go about finding u?

The Poisson Problem

Find u such that:

$$\begin{aligned} -\Delta u &= f, \quad \mathbf{x} \in \Omega \subset \mathbb{R}^2 \\ u &= 0, \quad \text{on } \partial\Omega. \end{aligned}$$

How do we go about finding u?

- Start by considering a variational formulation:

The Poisson Problem

Find u such that:

$$\begin{aligned} -\Delta u &= f, \quad \mathbf{x} \in \Omega \subset \mathbb{R}^2 \\ u &= 0, \quad \text{on } \partial\Omega. \end{aligned}$$

How do we go about finding u?

- Start by considering a variational formulation:

Find

$$u \in V = H_0^1(\Omega) = \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega \right\}$$

such that

$$\int_\Omega -\Delta u v \, d\Omega = \int_\Omega f v \, d\Omega, \;\; \forall \, v \in V.$$

The Poisson Problem

Find u such that:

$$\begin{aligned} -\Delta u &= f, \quad \mathbf{x} \in \Omega \subset \mathbb{R}^2 \\ u &= 0, \quad \text{on } \partial\Omega. \end{aligned}$$

How do we go about finding u?

- Start by considering a variational formulation:

After Integrating by parts:

$$\text{Find } u \in V = H_0^1(\Omega) = \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega \right\}$$

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

The Poisson Problem

Find u such that:

$$\begin{aligned} -\Delta u &= f, \quad \mathbf{x} \in \Omega \subset \mathbb{R}^2 \\ u &= 0, \quad \text{on } \partial\Omega. \end{aligned}$$

How do we go about finding u?

- Start by considering a variational formulation:

After Integrating by parts:

$$\text{Find } u \in V = H_0^1(\Omega) = \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega \right\}$$

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

- 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 Ω

Working with the problem domain Ω:

Let T_h be a triangulation of Ω

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

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 Ω

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

Triangulate Ω

Working with the problem domain Ω:
Let T_h be a triangulation of Ω

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

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 Ω

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

Triangulate Ω

Working with the problem domain Ω:
Let T_h be a triangulation of Ω

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

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 Ω

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

Triangulate Ω

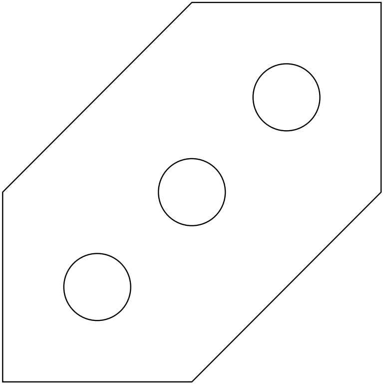
Working with the problem domain Ω:
Let T_h be a triangulation of Ω

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

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 Ω

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



Example Ω

Triangulate Ω

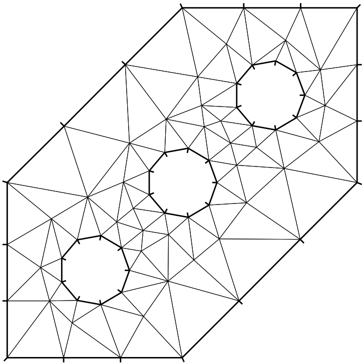
Working with the problem domain Ω:
Let T_h be a triangulation of Ω

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

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 Ω

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



Triangulation 1

Triangulate Ω

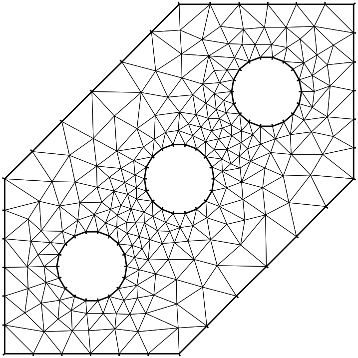
Working with the problem domain Ω:
Let T_h be a triangulation of Ω

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

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 Ω

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



Triangulation 2

Triangulate Ω

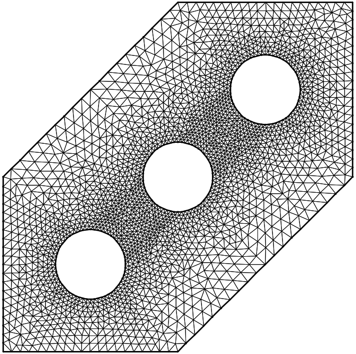
Working with the problem domain Ω:
Let T_h be a triangulation of Ω

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

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 Ω

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



Triangulation 3

Outline

Discrete Variational Formulation

Recall the variational formulation:

$$\text{Find } u \in V = H_0^1(\Omega) = \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega \right\}$$

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

Approximate with discrete variational formulation:

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

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

Here we choose the trial space (for u^h), and the test space (for v^h) to be V^h

Discrete Variational Formulation

Recall the variational formulation:

$$\text{Find } u \in V = H_0^1(\Omega) = \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega \right\}$$

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

Approximate with discrete variational formulation:

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

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

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

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

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

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

Quadratics:

$$\begin{aligned} \phi_1(\eta) &= 2(\eta - 1/2)(\eta - 1) \\ \phi_2(\eta) &= 4\eta(1 - \eta) \\ \phi_3(\eta) &= 2\eta(\eta - 1/2) \end{aligned}$$

Find a Basis for V^h

In one dimension on each element

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

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

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

Quadratics:

$$\begin{aligned} \phi_1(\eta) &= 2(\eta - 1/2)(\eta - 1) \\ \phi_2(\eta) &= 4\eta(1 - \eta) \\ \phi_3(\eta) &= 2\eta(\eta - 1/2) \end{aligned}$$

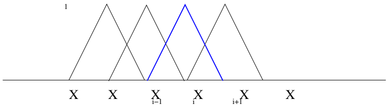
Find a Basis for V^h

In one dimension on each element

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

Linears:

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



Linear Basis (1-D)

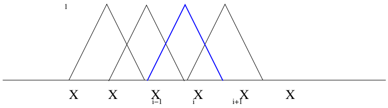
Find a Basis for V^h

In one dimension on each element

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

Linears:

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



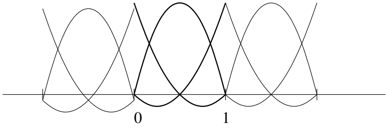
Linear Basis (1-D)

or

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

Quadratics:

$$\begin{aligned} \phi_1(\eta) &= 2(\eta - 1/2)(\eta - 1) \\ \phi_2(\eta) &= 4\eta(1 - \eta) \\ \phi_3(\eta) &= 2\eta(\eta - 1/2) \end{aligned}$$



Quadratic Basis (1-D)

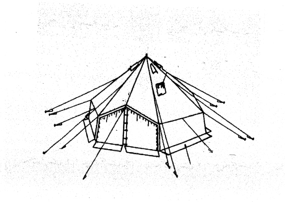
Find a Basis for V^h

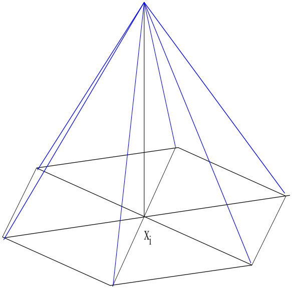
In two dimensions we use "Tent Functions." For example defined by

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

such that

$$\phi_i(\mathbf{x}_i) = 1 \quad \text{and} \quad \phi_i(\mathbf{x}_j) = 0 \quad \text{if } j \neq i$$





Note: All defined basis functions have **local support**

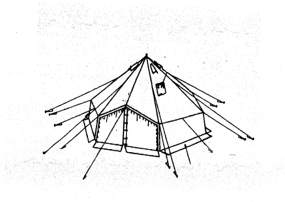
Find a Basis for V^h

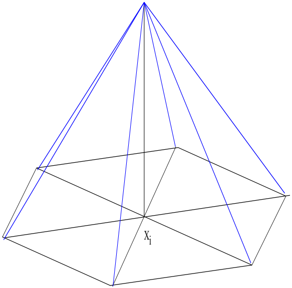
In two dimensions we use "Tent Functions." For example defined by

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

such that

$$\phi_i(\mathbf{x}_i) = 1 \quad \text{and} \quad \phi_i(\mathbf{x}_j) = 0 \quad \text{if } j \neq i$$





Note: All defined basis functions have **local support**

Assemble the Approximating System

Approximate

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

The approximating system with $v^h = \phi_i(\mathbf{x})$ becomes:

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

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

$$\Longrightarrow \quad \sum_{j=1}^{N} a_{ij} c_j = b_i$$

The Approximating System

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

$$A\mathbf{c} = \mathbf{b}$$

where

$$\begin{aligned} a_{ij} &= \int_\Omega \nabla\phi_j(\mathbf{x}) \cdot \nabla\phi_i(\mathbf{x})\, d\Omega \\ b_i &= \int_\Omega f\phi_i(\mathbf{x})\, d\Omega \end{aligned}$$

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:

$$A\mathbf{c} = \mathbf{b}$$

where

$$\begin{aligned} a_{ij} &= \int_\Omega \nabla\phi_j(\mathbf{x}) \cdot \nabla\phi_i(\mathbf{x})\, d\Omega \\ b_i &= \int_\Omega f\,\phi_i(\mathbf{x})\, d\Omega \end{aligned}$$

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:

$$A\mathbf{c} = \mathbf{b}$$

where

$$\begin{aligned} a_{ij} &= \int_\Omega \nabla\phi_j(\mathbf{x}) \cdot \nabla\phi_i(\mathbf{x})\, d\Omega \\ b_i &= \int_\Omega f\phi_i(\mathbf{x})\, d\Omega \end{aligned}$$

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:

$$A\mathbf{c} = \mathbf{b}$$

where

$$\begin{aligned} a_{ij} &= \int_\Omega \nabla\phi_j(\mathbf{x}) \cdot \nabla\phi_i(\mathbf{x})\, d\Omega \\ b_i &= \int_\Omega f\,\phi_i(\mathbf{x})\, d\Omega \end{aligned}$$

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:

$$A\mathbf{c} = \mathbf{b}$$

where

$$\begin{aligned} a_{ij} &= \int_\Omega \nabla\phi_j(\mathbf{x}) \cdot \nabla\phi_i(\mathbf{x})\, d\Omega \\ b_i &= \int_\Omega f\phi_i(\mathbf{x})\, d\Omega \end{aligned}$$

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

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

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

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

FreeFem++ for the Poisson Problem

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

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

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*

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

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

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

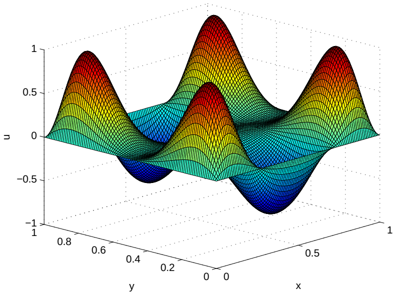
FreeFem++ for the Poisson Problem

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

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

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:

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

or, more flexible code can be written:

```
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:

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

or, more flexible code can be written:

```
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:

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

or, more flexible code can be written:

```
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]);
```

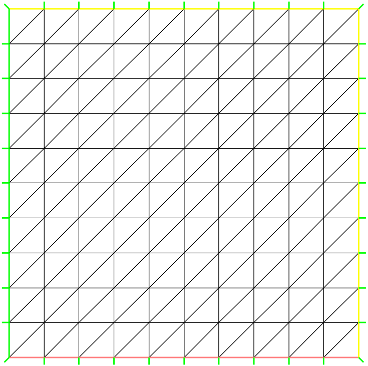
Build the mesh

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

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

or, more flexible code can be written:

```
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 $\mathcal{P}^1$ elements for u^h and v^h:

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

Declaring functions in V^h is easy:

```
Vh uh, vh;
```

We specify the right-hand side and boundary functions:

```
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 $\mathcal{P}^1$ elements for u^h and v^h:

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

Declaring functions in V^h is easy:

```
Vh uh, vh;
```

We specify the right-hand side and boundary functions:

```
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 $\mathcal{P}^1$ elements for u^h and v^h:

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

Declaring functions in V^h is easy:

```
Vh uh, vh;
```

We specify the right-hand side and boundary functions:

```
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 $\mathcal{P}^1$ elements for u^h and v^h:

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

Declaring functions in V^h is easy:

```
Vh uh, vh;
```

We specify the right-hand side and boundary functions:

```
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

```
int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))
```

and (f, v^h) as

```
int2d(Th)(f*vh)
```

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

```
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

```
int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))
```

and (f, v^h) as

```
int2d(Th)(f*vh)
```

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

```
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

```
int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))
```

and (f, v^h) as

```
int2d(Th)(f*vh)
```

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

```
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:

```
poisson;
```

Then we can plot the solution:

```
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);
```

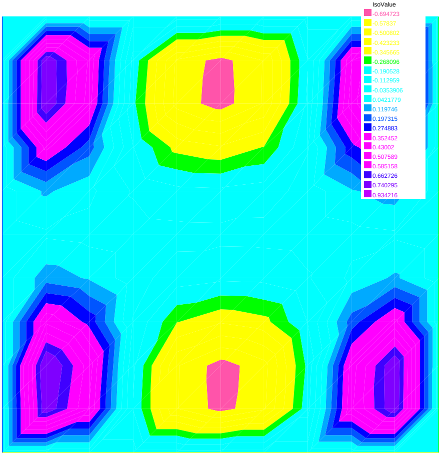
Solving the Problem and Viewing the Solution

The problem is solved by simply executing the problem statement:

```
poisson;
```

Then we can plot the solution:

```
plot(uh,fill=1,value=1);
```



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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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

```
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;
```

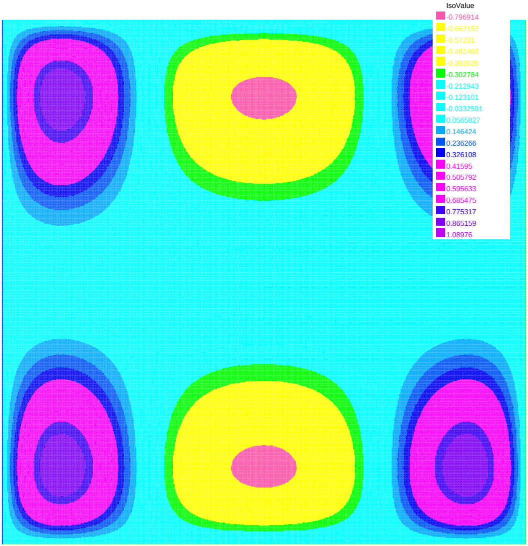
Convergence to the Exact Solution

As theory predicts, we have

$$\|u - u^h\|_0 \le Ch^2 \quad \text{and} \quad \|u - u^h\|_1 \le Ch$$

n	$\|u - u^h\|_0$	rate	$\|u - u^h\|_1$	rate
10	0.087719		2.43762	
20	0.024366	1.85	1.27541	0.93
40	0.006278	1.96	0.64626	0.98
80	0.001582	1.99	0.32425	1.00
160	0.000396	2.00	0.16227	1.00
320	0.000099	2.00	0.08115	1.00

Plot of Solution on Finest Mesh



Outline

Description of the Stokes Problem

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

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

Here

$$a(\mathbf{u}^h, \mathbf{v}^h) = \int_\Omega \nabla \mathbf{u}^h : \nabla \mathbf{v}^h \, d\Omega = \int_\Omega \left(u^h_{1,x} v^h_{1,x} + u^h_{2,x} v^h_{2,x} + u^h_{1,y} v^h_{1,y} + u^h_{2,y} v^h_{2,y} \right) d\Omega$$

and

$$b(q^h, \mathbf{u}^h) = \int_\Omega q^h \, div\, \mathbf{u}^h \, d\Omega = \int_\Omega q^h \left(u^h_{1,x} + u^h_{2,y} \right) \, d\Omega$$

Description of the Stokes Problem

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

$$\begin{aligned} a(\mathbf{u}^h, \mathbf{v}^h) + b(p^h, \mathbf{v}^h) &= (\mathbf{f}, \mathbf{v}^h) \quad \forall\, \mathbf{v}^h \in \mathbf{V}^h\,, \\ b(q^h, \mathbf{u}^h) &= 0 \qquad\quad\; \forall\, q^h \in Q^h\,. \end{aligned}$$

Here

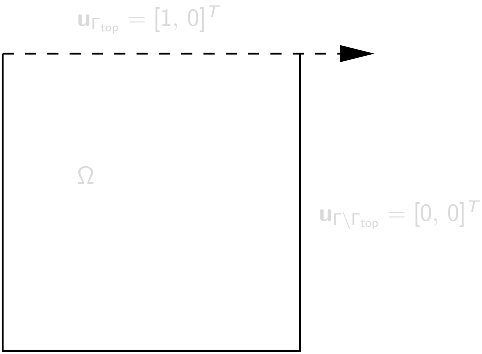
$$a(\mathbf{u}^h, \mathbf{v}^h) = \int_\Omega \nabla \mathbf{u}^h : \nabla \mathbf{v}^h \, d\Omega = \int_\Omega \left(u^h_{1,x} v^h_{1,x} + u^h_{2,x} v^h_{2,x} + u^h_{1,y} v^h_{1,y} + u^h_{2,y} v^h_{2,y} \right) d\Omega$$

and

$$b(q^h, \mathbf{u}^h) = \int_\Omega q^h \, div\, \mathbf{u}^h \, d\Omega = \int_\Omega q^h \left(u^h_{1,x} + u^h_{2,y} \right) \, d\Omega$$

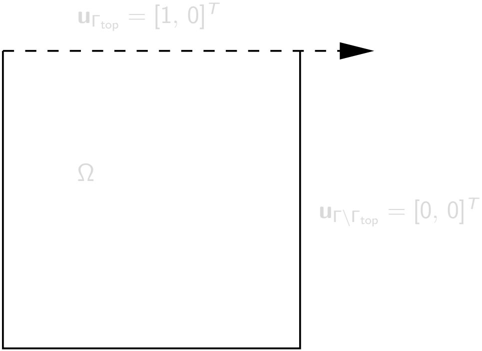
The Driven Cavity

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



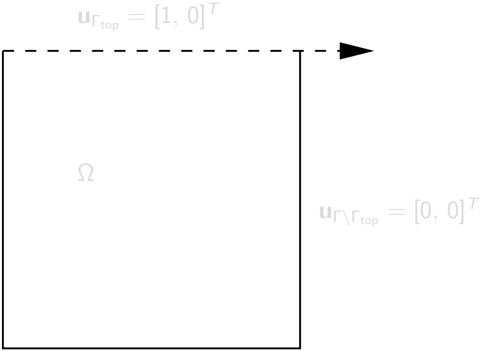
The Driven Cavity

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



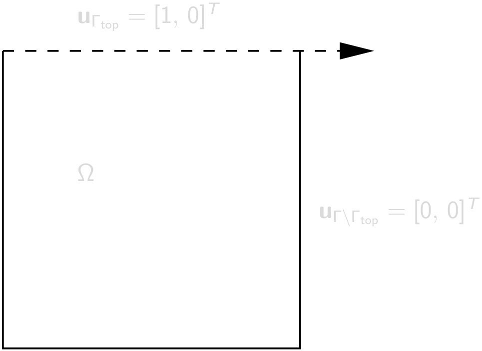
The Driven Cavity

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



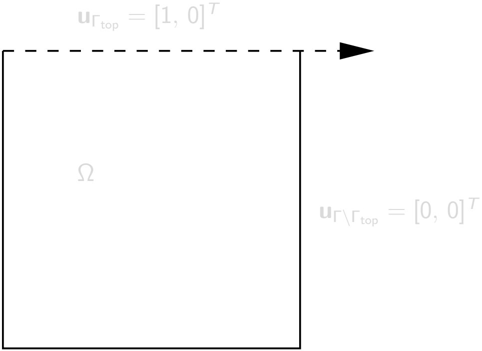
The Driven Cavity

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



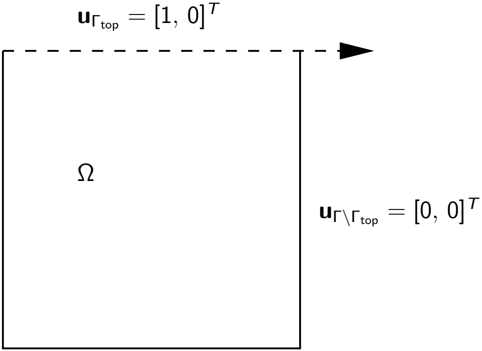
The Driven Cavity

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



The Driven Cavity

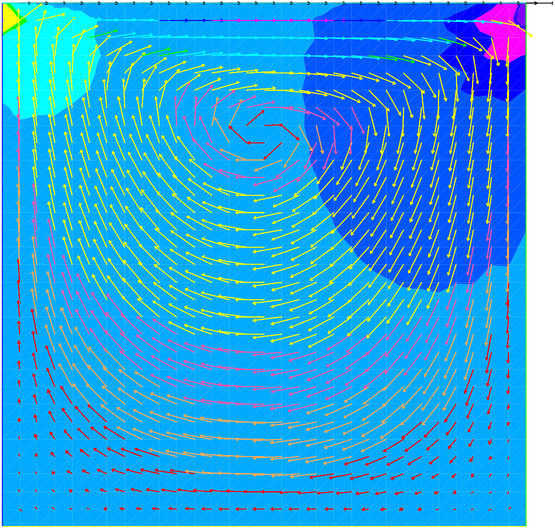
- A square cavity is filled with fluid
- The horizontal velocity at the top of the cavity is set to 1
- $\mathbf{f} = \mathbf{0}$



FreeFem++ code for Stokes Driven Cavity Problem

```
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);
```

Driven Cavity Problem Results



Mixed Finite Elements used for Stokes

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

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

One choice of $\mathbf{V}^h$ and Q^h that satisfies the condition is:

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

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

Warning: using elements that do not satisfy the mathematical framework can produce disastrous results!

Mixed Finite Elements used for Stokes

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

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

One choice of $\mathbf{V}^h$ and Q^h that satisfies the condition is:

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

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

Warning: using elements that do not satisfy the mathematical framework can produce disastrous results!

Mixed Finite Elements used for Stokes

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

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

One choice of $\mathbf{V}^h$ and Q^h that satisfies the condition is:

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

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

Warning: using elements that do not satisfy the mathematical framework can produce disastrous results!

An Example of Incompatible Elements

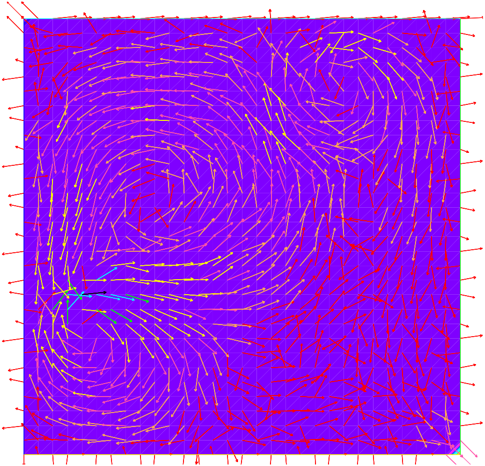
One choice of $\mathbf{V}^h$ and Q^h that does NOT satisfy the compatibility condition is:

```
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:

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



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

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

The Steady Navier-Stokes Equations

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

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

Linearization of the problem for use in a Newton Iteration: Given $(\mathbf{u}_0^h, p_0^h)$, for $i = 1, 2, \ldots,$ find $(\mathbf{u}_i^h, p_i^h)$ where

$$a(\mathbf{u}_i^h, \mathbf{v}^h) + b(p_i^h, \mathbf{v}^h) + (\mathbf{u}_i^h \cdot \nabla \mathbf{u}_{i-1}^h, \mathbf{v}^h) + (\mathbf{u}_i^h \cdot \nabla \mathbf{u}_{i-1}^h, \mathbf{v}^h) + b(q^h, \mathbf{u}_i^h) = (\mathbf{f}, \mathbf{v}^h) + (\mathbf{u}_{i-1}^h \cdot \nabla \mathbf{u}_{i-1}^h, \mathbf{v}^h) \quad \forall (\mathbf{v}^h, q^h) \in \mathbf{V}^h \times Q^h .$$

The Steady Navier-Stokes Equations

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

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

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

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

FreeFem++ code for Navier-Stokes Nonlinear Iteration

```
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;
}
```

Input/Output

Sample file manipulation:

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

Sample command-line input and output:

```
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;
}
```

Input/Output

Sample file manipulation:

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

Sample command-line input and output:

```
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;
}
```

Adaptive Mesh Refinement

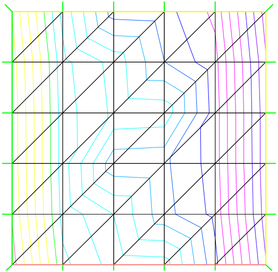
FreeFem++ has built-in mesh adaptivity routines.

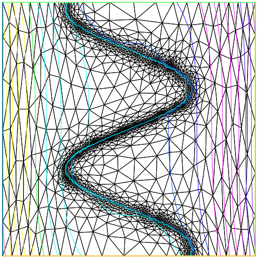
```
func f = 10.0*x^3+y^3
    +atan2(0.0001,sin(5.0*y)-2.0*x);
mesh Th=square(5,5,[-1+2*x,-1+2*y]);
fespace Vh(Th,P1);
Vh fh=f;
plot(fh);
for (int i=0;i<2;i++) {
    Th=adaptmesh(Th,fh);
    fh=f;
    plot(Th,fh);
}
```

Adaptive Mesh Refinement

`FreeFem++` has built-in mesh adaptivity routines.

```
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);
}
```





Scripting

FreeFem++ will allow for command line scripting:

```
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:

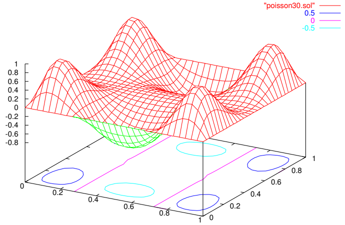
- 3D plotting
- LaTeX error reports
- Calling C or other code.

Scripting

FreeFem++ will allow for command line scripting:

```
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:

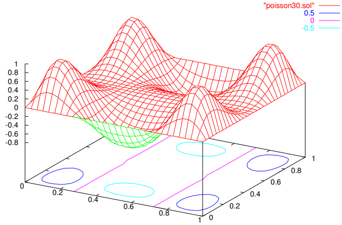
- 3D plotting
- LaTeX error reports
- Calling C or other code.

Scripting

`FreeFem++` will allow for command line scripting:

```
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.

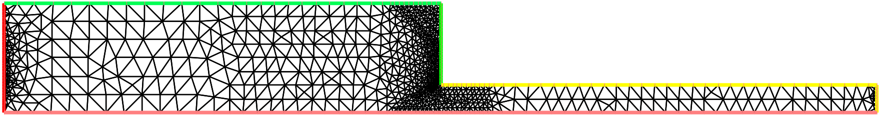
Advanced Meshing

Meshes can be created by parametrizations of the boundary:

```
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:

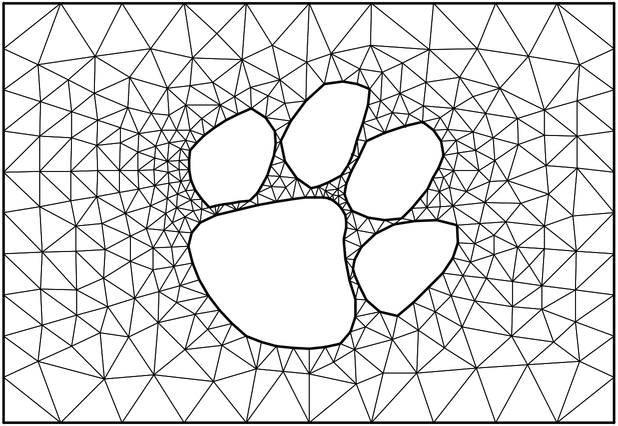


Advanced Meshing, part 2

You can even create really cool meshes:

Advanced Meshing, part 2

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.