Finite element programming by FreeFem++

Atsushi Suzuki[1]

[1]Cybermedia Center, Osaka University
atsushi.suzuki@cas.cmc.osaka-u.ac.jp

RIIT Tutorial, Kyushu Univ. 25, November 2016

Numerical simulation with finite element method

- mathematical modeling
- discretization of time for evolution problem
- discretization scheme for the space
  - mesh generation / adaptive mesh refinement
  - stiffness matrix from finite elements and variational formulation
  - linear solver $\Leftarrow$ `CG`, `GMRES`, direct solver: `UMFPACK`, `MUMPS`

  FreeFem++ provides vast amounts of tools
- nonlinear solver
- optimization solver

Outline I

Weak formulation for partial differential equations with 2nd order derivatives

- Poisson equation with mixed boundary conditions
- Stokes equations with mixed boundary conditions
- magneto-static problem with Dirichlet boundary conditions
- P1/P2 finite elements and numerical quadrature

Finite element stiffness matrix from weak form

- positivity of the matrix from coercivity of the bilinear form
- a penalty method to treat Dirichlet boundary data
- direct method
- CG / GMRES methods for symmetric/unsymmetric sparse matrices

Nonlinear finite element solution by Newton method

- stationary Navier-Stokes equations and a weak formulation
- differential calculus of nonlinear operator and Newton iteration

Outline II

Syntax and data structure of FreeFem++ language

- control flow
- data structure; array and FE object
- sparse matrix
- function macro
- procedure to compute SpMV for built-in iterative solver

Schwarz algorithm as preconditioner for global Krylov iteration

- overlapping subdomains and RAS/ASM
- 2-level algorithm with a coarse space

Time-dependent Navier-Stokes equations around a cylinder

- boundary conditions of incompressible flow around a cylinder

Thermal convection problem by Rayleigh-Bénard eqs.

- governing equations by Boussinesq approximation
- stationary solution by Newton method from numerical stationary solution

Outline

- Weak formulation for partial differential equations with 2nd order derivatives
  - Poisson equation with mixed boundary conditions
  - Stokes equations with mixed boundary conditions
  - magneto-static problem with Dirichlet boundary conditions
  - P1/P2 finite elements and numerical quadrature
- Finite element stiffness matrix from weak form
- Nonlinear finite element solution by Newton method
- Syntax and data structure of FreeFem++ language
- Schwarz algorithm as preconditioner for global Krylov iteration
- Time-dependent Navier-Stokes equations around a cylinder

Poisson equation with mixed B.C. and a weak formulation: 1/2

$\Omega \subset \mathbb{R}^2$, $\partial\Omega = \Gamma_D \cup \Gamma_N$

$$\begin{aligned} -\triangle u &= f \text{ in } \Omega, \\ u &= g \text{ on } \Gamma_D, \\ \frac{\partial u}{\partial n} &= h \text{ on } \Gamma_N. \end{aligned}$$

weak formulation

V : function space, $V(g) = \{u \in V\,;\, u = g \text{ on } \Gamma_D\}$.

$V = C^1(\Omega) \cap C^0(\bar{\Omega})$?

Find $u \in V(g)$ s.t.

$$\int_\Omega -\triangle u\, v dx = \int_\Omega f\, v dx \quad \forall v \in V(0)$$

Lemma (Gauss-Green's formula)

$u, v \in V$, $n = \begin{bmatrix} n_1 \\ n_2 \end{bmatrix}$: outer normal to $\partial\Omega$

$$\int_\Omega (\partial_i u) v\, dx = -\int_\Omega u \partial_i v\, dx + \int_{\partial\Omega} u\, n_i v\, ds\,.$$

Poisson equation with mixed B.C. and a weak formulation: 2/2

$$\int_\Omega (-\partial_1^2 - \partial_2^2) u\, v\, dx = \int_\Omega (\partial_1 u \partial_1 v + \partial_2 u \partial_2 v)\, dx - \int_{\partial\Omega} (\partial_1 u\, n_1 + \partial_2 u\, n_2) v\, ds$$

$$= \int_\Omega \nabla u \cdot \nabla v\, dx - \int_{\Gamma_D \cup \Gamma_N} \nabla u \cdot n\, v\, ds$$

$$v = 0 \text{ on } \Gamma_D \Rightarrow \quad = \int_\Omega \nabla u \cdot \nabla v\, dx - \int_{\Gamma_N} hv\, ds$$

Find $u \in V(g)$ s.t.

$$\int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f\, v dx + \int_{\Gamma_N} h\, v ds \quad \forall v \in V(0)$$

- $a(\cdot, \cdot) : V \times V \to \mathbb{R}$: bilinear form
- $F(\cdot) : V \to \mathbb{R}$: functional

Find $u \in V(g)$ s.t.

$$a(u, v) = F(v) \quad \forall v \in V(0)$$

FreeFem++ script to solve Poisson equation

finite element basis, span$[\varphi_1, \ldots, \varphi_N] = V_h \subset V$
$u_h \in V_h \;\Rightarrow\; u_h = \sum_{1\le i\le N} u_i \varphi_i$
Dirichlet data : $u(P_j) = g(P_j) \quad P_j \in \Gamma_D$
Find $u_h \in V_h(g)$ s.t.

$$\int_\Omega \nabla u_h \cdot \nabla v_h dx = \int_\Omega f\, v_h dx + \int_{\Gamma_N} h\, v_h ds \quad \forall v_h \in V_h(0).$$

▸ example1.edp ▸▸ varf+matrix

```
mesh Th=square(20,20);
fespace Vh(Th,P1);
Vh uh,vh;
func f=5.0/4.0*pi*pi*sin(pi*x)*sin(pi*y/2.0);
func g=sin(pi*x)*sin(pi*y/2.0);
func h=(-pi)/2.0*sin(pi*x);
solve poisson(uh,vh)=
   int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))
 - int2d(Th)(f*vh) - int1d(Th,1)(h*vh)
 + on(2,3,4,uh=g);          // boundary 1 : (x,0)
plot(uh);
```

Stokes equations and a weak formulation : 1/3

$\Omega = (0,1) \times (0,1)$

$$\begin{aligned} -2\nabla \cdot D(u) + \nabla p &= f \text{ in } \Omega \\ \nabla \cdot u &= 0 \text{ in } \Omega \\ u &= g \text{ on } \Gamma_D, \\ 2D(u)n - n\,p &= h \text{ on } \Gamma_N. \end{aligned}$$

strain rate tensor : $[D(u)]_{i\,j} = \frac{1}{2}(\partial_i u_j + \partial_j u_i)$.

- $V(g) = \{v \in H^1(\Omega)^2\,;\, v = g \text{ on } \Gamma_D\},\ V = V(0)$
- $Q = L^2(\Omega)$

weak formulation : Find $(u,p) \in V(g) \times Q$ s.t.

$$\begin{aligned} \int_\Omega 2D(u) : D(v)\,dx - \int_\Omega \nabla \cdot v\,p\,dx &= \int_\Omega f \cdot v\,dx + \int_{\Gamma_N} h \cdot v\,dx \quad \forall v \in V, \\ -\int_\Omega \nabla \cdot u\,q\,dx &= 0 \quad \forall q \in Q. \end{aligned}$$

Lemma (Gauss-Green's formula)

$u, v \in C^1(\Omega) \cap C^0(\bar{\Omega})$, n *: outer normal to* $\partial\Omega$

$$\int_\Omega (\partial_i u) v \, dx = -\int_\Omega u \partial_i v \, dx + \int_{\partial\Omega} u \, n_i v \, ds \, .$$

$$-2 \int_\Omega (\nabla \cdot D(u)) \cdot v \, dx =$$

$$\begin{aligned} -2 \int_\Omega \textstyle\sum_i \sum_j \partial_j \frac{1}{2}(\partial_i u_j + \partial_j u_i) v_i \, dx &= \int_\Omega \textstyle\sum_{i,j} (\partial_i u_j + \partial_j u_i) \partial_j v_i \, dx \\ &\quad - \int_{\partial\Omega} \textstyle\sum_{i,j} (\partial_i u_j + \partial_j u_i) n_j v_i \, ds \\ &= \int_\Omega 2D(u) : D(v) \, dx - \int_{\partial\Omega} 2D(u) \, n \cdot v \, ds \end{aligned}$$

from the symmetry of $D(u)$

$\sum_{i,j} (\partial_i u_j + \partial_j u_i) \partial_j v_i = \sum_{i,j} (\partial_i u_j + \partial_j u_i)(\partial_j v_i + \partial_i v_j)/2 = 2D(u) : D(v)$.

Stokes equations and a weak formulation : 3/3

$$\begin{aligned}\int_\Omega \textstyle\sum_i(\partial_i p)\, v_i\, dx &= -\int_\Omega \textstyle\sum_i p\partial_i v_i\, dx + \int_{\partial\Omega}\textstyle\sum_i p\, n_i v_i \\ &= -\int_\Omega p\nabla\cdot v + \int_{\partial\Omega} p\, n\cdot v\end{aligned}$$

On the boundary $\partial\Omega = \Gamma_D \cup \Gamma_N$,

$$\begin{aligned}&\int_{\Gamma_D\cup\Gamma_N}(2D(u)n - n\, p)\cdot v\, ds \\ &\qquad = \int_{\Gamma_D}(2D(u)n - n\, p)\cdot v\, ds + \int_{\Gamma_N}(2D(u)n - n\, p)\cdot v\, ds \\ &\qquad = \int_{\Gamma_N} h\cdot v\, ds. \quad \Leftarrow V = \{v\in H^1(\Omega)^2\,;\, v = 0 \text{ on } \partial\Omega\}.\end{aligned}$$

Remark

$-2[\nabla\cdot D(u)]_i = -\sum_j \partial_j(\partial_i u_j + \partial_j u_i) = -\sum_j \partial_j^2 u_i = -[\triangle u]_i.$

FreeFem++ script to solve Stokes equations by P2/P1

Find $(u, p) \in V_h(g) \times Q_h$ s.t.

$a(u, v) + b(v, p) + b(u, q) = (f, v) \quad \forall (v, q) \in V_h \times Q_h$.

▸ example2.edp

```
fespace Vh(Th,P2),Qh(Th,P1);
func f1=5.0/8.0*pi*pi*sin(pi*x)*sin(pi*y/2.0)+2.0*x;
func f2=5.0/4.0*pi*pi*cos(pi*x)*cos(pi*y/2.0)+2.0*y;
func g1=sin(pi*x)*sin(pi*y/2.0)/2.0;
func g2=cos(pi*x)*cos(pi*y/2.0);
func h1=3.0/4.0*pi*sin(pi*x)*cos(pi*y/2.0);
func h2=pi*cos(pi*x)*sin(pi*y/2.0)+x*x+y*y;
Vh u1,u2,v1,v2; Qh p,q;
macro d12(u1,u2)  (dy(u1) + dx(u2))/2.0 //
real epsln=1.0e-6;
solve stokes(u1,u2,p, v1,v2,q) =
int2d(Th)(2.0*(dx(u1)*dx(v1)+dy(u2)*dy(v2)+
               2.0*d12(u1,u2)*d12(v1,v2))
         -p*dx(v1)-p*dy(v2)-dx(u1)*q-dy(u2)*q)
- int2d(Th)(f1*v1+f2*v1) - int1d(Th,1)(h1*v1+h2*v1)
+ on(2,3,4,u1=g1,u2=g2);
plot([u1,u2],p,wait=1,value=true,coef=0.1);
```

stationary Navier-Stokes equations and a weak formulation

$\Omega = (0,1) \times (0,1)$

$$-2\nu \nabla \cdot D(u) + u \cdot \nabla u + \nabla p = f \text{ in } \Omega$$
$$\nabla \cdot u = 0 \text{ in } \Omega$$
$$u = g \text{ on } \partial\Omega$$

- $V(g) = \{v \in H^1(\Omega)^2\,;\, v = g \text{ on } \partial\Omega\},\ \ V = V(0)$
- $Q = L^2_0(\Omega) = \{p \in L^2(\Omega)\,;\, \int_\Omega p\,dx = 0\}$

outflow

bi/tri-linear forms and weak formulation :

$$a(u,v) = \int_\Omega 2\nu D(u) : D(v)\,dx \quad u, v \in H^1(\Omega)^2$$

$$a_1(u,v,w) = \int_\Omega (u \cdot \nabla v) \cdot w \quad u, v, w \in H^1(\Omega)^2$$

$$b(v,p) = -\int_\Omega \nabla \cdot v\, p\,dx \quad v \in H^1(\Omega)^2,\ p \in L^2(\Omega)$$

Find $(u,p) \in V(g) \times Q$ s.t.

$$a(u,v) + a_1(u,u,v) + b(v,p) = (f,v) \quad \forall v \in V,$$
$$b(u,q) = 0 \quad \forall q \in Q.$$

magneto-static problem with Dirichlet b.c. : 1/2

constraints on the external force: $\nabla \cdot f = 0$ in $\Omega \subset \mathbb{R}^3$.

$$\begin{aligned} \nabla \times (\nabla \times u) &= f && \text{in } \Omega, \\ \nabla \cdot u &= 0 && \text{in } \Omega, \\ u \times n &= 0 && \text{on } \partial\Omega. \end{aligned}$$

$H_0(\mathrm{curl}\,;\,\Omega) = \{u \in L^2(\Omega)^3\,;\, \nabla \times u \in L^2(\Omega)^3\,;\, u \times n = 0\}$

find $(u, p) \in H_0(\mathrm{curl}\,;\,\Omega) \times H^1_0(\Omega)$

$$\begin{aligned} (\nabla \times u, \nabla \times v) + (v, \nabla p) &= (f, v) && \forall v \in H_0(\mathrm{curl}\,;\,\Omega) \\ (u, \nabla q) &= 0 && \forall v \in H^1_0(\Omega) \end{aligned}$$

has a unique solution.

- $\nabla \cdot f = 0 \Rightarrow p = 0$.
- $(\nabla\times\cdot, \nabla\times\cdot)$: coercive on W.
  $$W = H_0(\mathrm{curl}\,;\,\Omega) \cap \{u \in H(\mathrm{div};\Omega)\,;\, \mathrm{div}\, u = 0\}.$$
- $H_0(\mathrm{curl}\,;\,\Omega) = \mathrm{grad} H^1_0(\Omega) \oplus W$.

magneto-static problem with Dirichlet b.c. : 2/2

vector valued Sobolev space:
$H(\mathrm{curl}\,;\,\Omega) = \{u \in L^2(\Omega)^3\,;\,\nabla \times u \in L^2(\Omega)^3\}$.
with essential boundary conditions on $\partial\Omega$:
$H_0(\mathrm{curl}\,;\,\Omega) = \{u \in L^2(\Omega)^3\,;\,\nabla \times u \in L^2(\Omega)^3\,;\,u \times n = 0\}$.

$$\int_\Omega \nabla \times u \cdot v dx = \int_\Omega \left(\begin{bmatrix} \partial_1 \\ \partial_2 \\ \partial_3 \end{bmatrix} \times \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \right) \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} dx$$

$$= \int_\Omega \begin{pmatrix} (\partial_2 u_3 - \partial_3 u_2) v_1 + \\ (\partial_3 u_1 - \partial_1 u_3) v_2 + \\ (\partial_1 u_2 - \partial_2 u_1) v_3 \end{pmatrix} dx$$

$$= -\int_\Omega \begin{pmatrix} (u_3 \partial_2 v_1 - u_2 \partial_3 v_1) + \\ (u_1 \partial_3 v_2 - u_3 \partial_1 v_2) + \\ (u_2 \partial_1 v_3 - u_1 \partial_2 v_3) \end{pmatrix} dx + \int_{\partial\Omega} \begin{pmatrix} (n_2 u_3 - n_3 u_2) v_1 + \\ (n_3 u_1 - n_1 u_3) v_2 + \\ (n_1 u_2 - n_2 u_1) v_3 \end{pmatrix} ds$$

$$= \int_\Omega u \cdot \nabla \times v dx + \int_{\partial\Omega} (n \times u) \cdot v ds$$

stiffness matrix of magneto-static problem by FreeFem++

```
load "msh3"
load "Dissection"
defaulttoDissection;
mesh3 Th=cube(20,20,20);
fespace VQh(Th, [Edge03d, P1]); // Nedelec element
VQh [u1, u2, u3, p], [v1, v2, v3, q];
varf aa([u1, u2, u3, p], [v1, v2, v3, q]) =
  int3d(Th)((dy(u3)-dz(u2)) * (dy(v3)-dz(v2)) +
            (dz(u1)-dx(u3)) * (dz(v1)-dx(v3)) +
            (dx(u2)-dy(u1)) * (dx(v2)-dy(v1)) +
            dx(p)*v1+dy(p)*v2+dz(p)*v3 +
            dx(q)*u1+dy(q)*u2+dz(q)*u3);
matrix A = aa(VQh, VQh, solver=sparsesolver,
              tolpivot=1.0e-2,strategy=102);
```

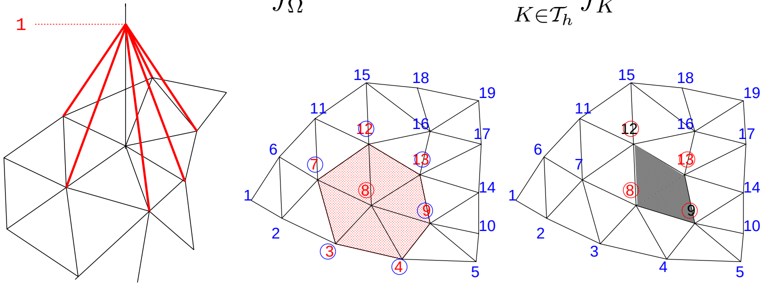
Nédélec element of degree 0 and P1

$$N_0(K) = (P_0(K))^3 \oplus [x \times (P_0(K))^3], \qquad P_1(K)$$

```
fespace VQh(Th,[Edge03d,P1]);
```

$\mathcal{T}_h$: triangulation of a domain Ω, triangular element $K \in \mathcal{T}_h$
piecewise linear element : $\varphi_i|_K(x_1, x_2) = a_0 + a_1 x_1 + a_2 x_2$
$\varphi_i|_K(P_j) = \delta_{i\,j}$

$$[A]_{i\,j} = a(\varphi_j, \varphi_i) = \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, dx = \sum_{K \in \mathcal{T}_h} \int_K \nabla\varphi_j \cdot \nabla\varphi_i \, dx.$$



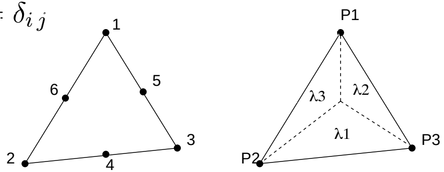
A : sparse matrix, CRS (Compressed Row Storage) format to store

P2 finite element

$\mathcal{T}_h$: triangulation of a domain Ω, triangular element $K \in \mathcal{T}_h$
piecewise quadratic element : 6 DOF on element K.

$\varphi_i|_K(x_1, x_2) = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_1^2 + a_4 x_1 x_2 + a_5 x_2^2$

$\varphi_i|_K(P_j) = \delta_{i\,j}$



by using area coordinates $\{\lambda_1, \lambda_2, \lambda_3\},\ \lambda_1 + \lambda_2 + \lambda_3 = 1$.

$$\begin{pmatrix}\varphi_1\\ \varphi_2\\ \varphi_3\\ \varphi_4\\ \varphi_5\\ \varphi_6\end{pmatrix} = \begin{pmatrix}1 & & & & -1 & -1\\ & 1 & & -1 & & -1\\ & & 1 & -1 & -1 & \\ & & & 4 & & \\ & & & & 4 & \\ & & & & & 4\end{pmatrix}\begin{pmatrix}\lambda_1^2\\ \lambda_2^2\\ \lambda_3^2\\ \lambda_2\lambda_3\\ \lambda_3\lambda_1\\ \lambda_1\lambda_2\end{pmatrix} = \begin{pmatrix}\lambda_1(2\lambda_1 - 1)\\ \lambda_2(2\lambda_2 - 1)\\ \lambda_3(2\lambda_3 - 1)\\ 4\lambda_2\lambda_3\\ 4\lambda_3\lambda_1\\ 4\lambda_1\lambda_2\end{pmatrix}$$

```
fespace Vh(Th,P2);
```

numerical integration

Numerical quadrature:

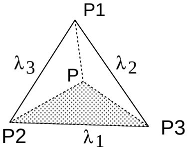
$\{P_i\}_{i\leq i\leq m}$: integration points in K, $\{\omega_i\}_{i\leq i\leq m}$: weights

$$|u-u_h|^2_{0,\Omega}=\sum_{K\in\mathcal{T}_h}\int_K |u-u_h|^2dx \sim \sum_{K\in\mathcal{T}_h}\sum_{i=1}^{m}|(u-u_h)(P_i)|^2\omega_i$$

formula : degree 5, 7 points, qf5pT,

P.C. Hammer, O.J. Marlowe, A.H. Stroud [1956]

area coordinates $\{\lambda_i\}_{i=1}^3$	weight	
$(\frac{1}{3},\frac{1}{3},\frac{1}{3})$	$\frac{9}{40}\lvert K\rvert$	$\times 1$
$(\frac{6-\sqrt{15}}{21},\frac{6-\sqrt{15}}{21},\frac{9+2\sqrt{15}}{21})$	$\frac{155-\sqrt{15}}{1200}\lvert K\rvert$	$\times 3$
$(\frac{6+\sqrt{15}}{21},\frac{6+\sqrt{15}}{21},\frac{9-2\sqrt{15}}{21})$	$\frac{155+\sqrt{15}}{1200}\lvert K\rvert$	$\times 3$



Remark

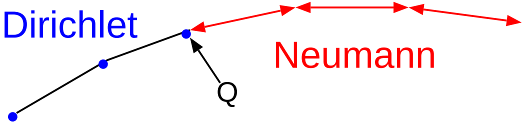
it is not good idea to use interpolation of continuous function to finite element space, for verification of convergence order.

$|\Pi_h u-u_h|_{1,\Omega}$ may be smaller (in extreme cases, super convergence)

treatment of Neumann data around mixed boundary

Neumann data is evaluated by line integral with FEM basis φ_i.

$$\int_{\Gamma_N} h\,\varphi_i\,ds$$



For given discrete Neumann data, h is interpolated in FEM space, $h = \sum_j h_j \varphi_j|_{\Gamma_N}$,

$$\sum_j h_j \int_{\Gamma_N} \varphi_j \varphi_i\,ds.$$

On the node $Q \in \bar{\Gamma}_D \cap \bar{\Gamma}_N$, both Dirichlet and Neumann are necessary.

advantages of finite element formulation : 1/2

- weak formulation is obtained by integration by part with clear description on the boundary
- Dirichlet boundary condition, called as essential boundary condition, is embedded in a functional space
- Neumann boundary condition, called as natural boundary condition, is treated with surface/line integral by Gauss-Green's formula
- computation of local stiffness matrix is performed by numerical quadrature formula

Outline

Weak formulation for partial differential equations with 2nd order derivatives

Finite element stiffness matrix from weak form
- positivity of the matrix from coercivity of the bilinear form
- a penalty method to treat Dirichlet boundary data
- direct method
- CG / GMRES methods for symmetric/unsymmetric sparse matrices

Nonlinear finite element solution by Newton method

Syntax and data structure of FreeFem++ language

Schwarz algorithm as preconditioner for global Krylov iteration

Time-dependent Navier-Stokes equations around a cylinder

discretization and matrix formulation : 1/2

finite element basis, span$[\varphi_1, \dots, \varphi_N] = V_h \subset V$
$u_h \in V_h \;\Rightarrow\; u_h = \sum_{1 \leq i \leq N} u_i \varphi_i$
finite element nodes $\{P_j\}_{j=1}^N$, $\varphi_i(P_j) = \delta_{i\,j}$ Lagrange element
$\Lambda_D \subset \Lambda = \{1, \dots, N\}$: index of node on the Dirichlet boundary

$$V_h(g) = \{u_h \in V_h \,;\, u_h = \sum u_i \varphi_i, u_k = g_k \; (k \in \Lambda_D)\}$$

Find $u_h \in V_h(g)$ s.t.

$$a(u_h, v_h) = F(v_h) \quad \forall v_h \in V_h(0).$$

Find $\{u_j\}, u_k = g_k(k \in \Lambda_D)$ s.t.

$$a(\sum_j u_j \varphi_j, \sum_i v_i \varphi_i) = F(\sum_i v_i \varphi_i) \; \forall \{v_i\}, v_k = 0 (k \in \Lambda_D)$$

Find $\{u_j\}_{j \in \Lambda}$s.t.

$$\begin{aligned} \sum_j a(\varphi_j, \varphi_i) u_j &= F(\varphi_i) & \forall i \in \Lambda \setminus \Lambda_D \\ u_k &= g_k & \forall k \in \Lambda_D \end{aligned}$$

discretization and matrix formulation : 2/2

Find $\{u_j\}_{j\in\Lambda\setminus\Lambda_D}$ s.t.

$$\sum_{j\in\Lambda\setminus\Lambda_D} a(\varphi_j,\varphi_i)u_j = F(\varphi_i) - \sum_{k\in\Lambda_D} a(\varphi_k,\varphi_i)g_k \quad \forall i\in\Lambda\setminus\Lambda_D$$

$$A=\{a(\varphi_j,\varphi_i)\}_{i,j\in\Lambda\setminus\Lambda_D} \qquad A\in\mathbb{R}^{n\times n},\ f\in\mathbb{R}^n,\ n=\#(\Lambda\setminus\Lambda_D)$$

Lemma

A *: positive definite (coercive)* $\Leftrightarrow (Au,u)>0\ \forall u\neq 0$
$\Rightarrow\ Au=f$ *has a unique solution.*

A : bijective

- injective: $Au=0,\ 0=(Au,u)>0\ \Rightarrow u=0$.
- surjective:
  $\mathbb{R}^n=\mathrm{Im}A\oplus(\mathrm{Im}A)^\perp$, $u\in(\mathrm{Im}A)^\perp\Rightarrow(Av,u)=0\ \forall v\in\mathbb{R}^n$
  by putting $v=u$, $0=(Au,u)\ \Rightarrow u=0$
  $(\mathrm{Im}A)^\perp=\{0\}\Rightarrow\mathrm{Im}A=\mathbb{R}^n$.

unymmetric $\Rightarrow$ solution by LDU-factorization, GMRES method
symmetric $\Rightarrow$ solution by LDL^T-factorization, CG method

Sobolev space

P1 element element space does not belong to $C^1(\Omega)$.

$V = H^1(\Omega)$,

$$(u,v) = \int_\Omega u\,v + \nabla u \cdot \nabla v, \quad ||u||_1^2 = (u,u) < +\infty$$

$$||u||_0^2 = \int_\Omega u\,u,$$

$$|u|_1^2 = \int_\Omega \nabla u \cdot \nabla u.$$

$H_0^1 = \{u \in H^1(\Omega)\,;\, u = 0 \text{ on } \partial\Omega\}$.

Lemma (Poincaré's inequality)

$\exists C(\Omega)\ u \in H_0^1 \ \Rightarrow\ ||u||_0 \leq C(\Omega)|u|_1$.

$a(u,v) = \int_\Omega \nabla u \cdot \nabla v$ is coercive on V.

$a(u,u) = |u|_1^2 \geq \dfrac{1}{C^2}||u||_0^2$.

FreeFem++ script to solve Poisson eq. using matrix

Find $u_h \in V_h(g)$ s.t. $a(u_h, v_h) = F(v_h)\ \forall v_h \in V_h(0)$.

▸ example3.edp ▸▸ solve

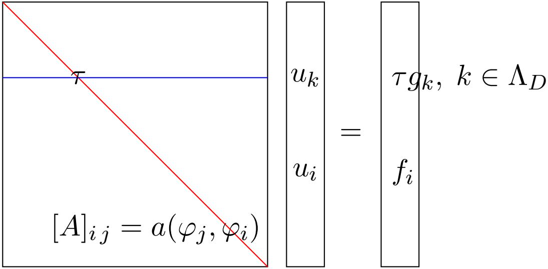
```
Vh u,v;
varf aa(u,v)=int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))
            +on(2,3,4,u=g);
varf external(u,v)=int2d(Th)(f*v)+int1d(Th,1)(h*v)
            +on(2,3,4,u=g);
real tgv=1.0e+30;
matrix A = aa(Vh,Vh,tgv=tgv,solver=CG);
real[int] ff = external(0,Vh,tgv=tgv);
u[] = A^-1 * ff;     // u : fem unknown, u[] : vector
plot(u);
```

useful liner solver; `solver=`

`CG`	iterative solver for SPD matrix
`GMRES`	iterative solver for nonsingular matrix
`UMFPACK`	direct solver for nonsingular matrix
`sparsesolver`	other solvers called by dynamic link

penalty method to solve inhomogeneous Dirichlet problem

modification of diagonal entries of A where index $k \in \Lambda_D$
penalization parameter $\tau = 1/\varepsilon$; `tgv`



$$\tau u_k + \sum_{j \neq k} a_{k\,j} u_j = \tau g_k \ \Leftrightarrow\ u_k - g_k = \varepsilon(-\sum_{j \neq k} a_{k\,j} u_j),$$

$$\sum_j a_{i\,j} u_j = f_i \qquad \forall i \in \{1, \ldots, N\} \setminus \Lambda_D.$$

keeping symmetry of the matrix without changing index numbering.

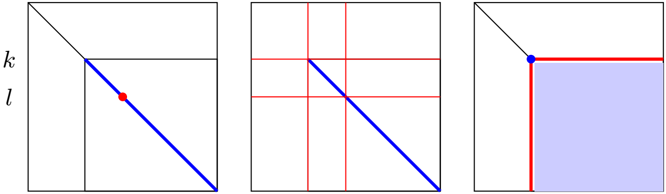
LDL^T/LDU factorization

$$\begin{bmatrix} a_{11} & \beta_1^T \\ \alpha_1 & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \alpha_1 a_{11}^{-1} & S_{22} \end{bmatrix} \begin{bmatrix} a_{11} & \beta_1^T \\ 0 & I_2 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 \\ \alpha_1 a_{11}^{-1} & S_{22} \end{bmatrix} \begin{bmatrix} a_{11} & 0 \\ 0 & I_2 \end{bmatrix} \begin{bmatrix} 1 & a_{11}^{-1}\beta_1^T \\ 0 & I_2 \end{bmatrix}$$

Schur complement $S_{22} = A_{22} - \alpha_1 a_{11}^{-1} \beta_1^T$ rank-1 update

LDU-factorization with symmetric pivoting : $A = \Pi^T \, LDU \, \Pi$

do $k = 1, \cdots, N$
  find $k < l \leq n$ max$|A(l,l)|$,
  exchange rows and columns : $A(k,*) \leftrightarrow A(l,*)$, $A(*,k) \leftrightarrow A(*,l)$.
  rank-1 update



conjugate gradient method

$$A\vec{u} = \vec{f}.$$

preconditioner $Q \sim A^{-1}$

Krylov subsp. : $K_n(Q\vec{r}^{\,0}, QA)$=span$[Q\vec{r}^{\,0}, QAQ\vec{r}^{\,0}, \dots, (QA)^{n-1}Q\vec{r}^{\,0}]$

Find $\vec{u}^{\,n} \in K_n(Q\vec{r}^{\,0}, QA) + \vec{u}^{\,0}$ s.t.

$$(A\vec{u}^{\,n} - \vec{f}, \vec{v}) = 0 \;\; \forall \vec{v} \in K_n(Q\vec{r}^{\,0}, QA).$$

Preconditioned CG method

$\vec{u}^{\,0}$: initial step for CG.

$\vec{r}^{\,0} = \vec{f} - A\vec{u}^{\,0}$

$\vec{p}^{\,0} = Q\vec{r}^{\,0}$.

loop $n = 0, 1, \dots$

$\alpha_n = (Q\vec{r}^{\,n}, \vec{r}^{\,n})/(A\vec{p}^{\,n}, \vec{p}^{\,n})$,

$\vec{u}^{\,n+1} = \vec{u}^{\,n} + \alpha_n \vec{p}^{\,n}$,

$\vec{r}^{\,n+1} = \vec{r}^{\,n} - \alpha_n A\vec{p}^{\,n}$,

if $||\vec{r}^{\,n+1}|| < \epsilon$ exit loop.

$\beta_n = (Q\vec{r}^{\,n+1}, \vec{r}^{\,n+1})/(Q\vec{r}^{\,n}, \vec{r}^{\,n})$,

$\vec{p}^{\,n+1} = Q\vec{r}^{\,n+1} + \beta_n \vec{p}^{\,n}$.

```
LinearCG(opA,u,f,precon=opQ,nbiter=100,eps=1.0e-10)
```

GMRES method : 1/2

Krylov subspace : $K_n(\vec{r}^{\,0}, A) = \text{span}[\vec{r}^{\,0}, A\vec{r}^{\,0}, \dots, A^{n-1}\vec{r}^{\,0}]$

Find $\vec{u}^{\,n} \in K_n(\vec{r}^{\,0}, A) + \vec{u}^{\,0}$ s.t.

$$||A\vec{u}^{\,n} - \vec{f}|| \leq ||A\vec{v}^{\,n} - \vec{f}|| \;\; \forall \vec{v} \in K_n(\vec{r}^{\,0}, A) + \vec{u}^{\,0}.$$

V_m : Arnoldi basis generated by Gram-Schmidt orthogonization for Krylov vectors.

$$\vec{u} = V_m\vec{y},\; \vec{y} \in \mathbb{R}^m \quad \begin{aligned} J(\vec{y}) &:= ||AV_m\vec{y} - \vec{r}_0|| \\ &= ||V_{m+1}^T(AV_m\vec{y} - \vec{r}_0)|| \\ &= ||(V_{m+1}^T AV_m)\vec{y} - (V_{m+1}^T\vec{r}_0)|| \\ &= ||\bar{H}_m\vec{y} - \beta\vec{e}_1||\,. \qquad (\beta = ||\vec{r}_0||) \end{aligned}$$

$$\text{Find } \vec{y} \in \mathbb{R}^m \quad J(\vec{y}) \leq J(\vec{z}) \quad \forall \vec{z} \in \mathbb{R}^m\,.$$

minimization problem with Hessenberg matrix $\bar{H}_m \in R^{(m+1)\times m}$ is solved by Givens rotation.

```
LinearGMRES(opA,u,f,precon=opQ,nbiter=100,
                                   eps=1.0e-10)
```

GMRES method : 2/2

Arnoldi method (Gram-Schmidt method on Krylov subspace)

$$||\vec{v}_1|| = 1; \text{ do } j = 1, 2, \ldots, m$$

$$\text{do } i = 1, 2, \ldots, j$$

$$\vec{w}_j := A\vec{v}_j - \textstyle\sum_{1 \le i \le j} h_{i\,j}\vec{v}_i\,, \quad h_{i\,j} := (A\,\vec{v}_j, \vec{v}_i)$$

$$\vec{v}_{j+1} := \vec{w}_j / h_{j+1\,j}\,, \quad h_{j+1\,j} := ||\vec{w}_j||$$

Givens rotation matrices $\Omega_i \in \mathbb{R}^{(m+1)\times(m+1)}$

$$\Omega_i := \begin{bmatrix} I_{i-1} & & & \\ & c_i & s_i & \\ & -s_i & c_i & \\ & & & I_{m-i} \end{bmatrix}, \; c_1 := \frac{h_{1\,1}}{\sqrt{h_{1\,1}^2 + h_{2\,2}^2}}\,,\; s_1 := \frac{h_{2\,1}}{\sqrt{h_{1\,1}^2 + h_{2\,2}^2}}\,.$$

$Q_m := \Omega_m \Omega_{m-1} \cdots \Omega_1 \in \mathbb{R}^{(m+1)\times(m+1)}$,

$\bar{R}_m := Q_m \bar{H}_m$: upper triangular,

$\bar{g}_m := Q_m(\beta e_1) = [\,\gamma_1\ \gamma_2\ \cdots\ \gamma_{m+1}\,]^T$,

$\bar{R}_m := \begin{bmatrix} R_m \\ 0\ \cdots\ 0 \end{bmatrix}$ ($R_m \in \mathbb{R}^{m\times m}$), $\bar{g}_m := \begin{bmatrix} g_m \\ \gamma_{m+1} \end{bmatrix}$ ($g_m \in \mathbb{R}^m$).

min$||\beta e_1 - \bar{H}_m y||$ = min$||\bar{g}_m - \bar{R}_m y|| = |\gamma_{m+1}| = |s_1 s_2 \cdots s_m|\beta\,.$

$y_m = R_m^{-1} g_m$ attains the minimum.

Remark : $\exists R_m^{-1}$ $(1 \le m \le M)$ for all non-singular matrix A.

advantages of finite element formulation 2/2

- solvability of linear system is inherited from solvability of continuous weak formulation

better to learn for efficient computation

- treatment of Dirichlet boundary conditions in FreeFem++ with explicit usage of matrix and linear solver
- Direct solver like UMFPACK is efficient for 2D problems, but for 3D iterative solvers such as CG/GMERES with good preconditioner (e.g. additive Schwarz method) are necessary.

Outline

- Weak formulation for partial differential equations with 2nd order derivatives
- Finite element stiffness matrix from weak form
- Nonlinear finite element solution by Newton method
  - stationary Navier-Stokes equations and a weak formulation
  - differential calculus of nonlinear operator and Newton iteration
- Syntax and data structure of FreeFem++ language
- Schwarz algorithm as preconditioner for global Krylov iteration
- Time-dependent Navier-Stokes equations around a cylinder
- Thermal convection problem by Rayleigh-Bénard eqs.

stationary Navier-Stokes equations and a weak formulation

$\Omega = (0,1) \times (0,1)$

$$-2\nu\nabla \cdot D(u) + u \cdot \nabla u + \nabla p = f \text{ in } \Omega$$
$$\nabla \cdot u = 0 \text{ in } \Omega$$
$$u = g \text{ on } \partial\Omega$$

- $V(g) = \{v \in H^1(\Omega)^2\,;\, v = g \text{ on } \partial\Omega\},\ V = V(0)$
- $Q = L^2_0(\Omega) = \{p \in L^2(\Omega)\,;\, \int_\Omega p\,dx = 0\}$

outflow

bi/tri-linear forms and weak formulation :

$$a(u,v) = \int_\Omega 2\nu D(u) : D(v)\,dx \quad u,v \in H^1(\Omega)^2$$

$$a_1(u,v,w) = \int_\Omega (u \cdot \nabla v) \cdot w \quad u,v,w \in H^1(\Omega)^2$$

$$b(v,p) = -\int_\Omega \nabla \cdot v\,p\,dx \quad v \in H^1(\Omega)^2,\ p \in L^2(\Omega)$$

Find $(u,p) \in V(g) \times Q$ s.t.

$$a(u,v) + a_1(u,u,v) + b(v,p) = (f,v) \quad \forall v \in V,$$
$$b(u,q) = 0 \quad \forall q \in Q.$$

nonlinear system of the stationary solution

$A(u,p\,;\,v,q) = a(u,v) + a_1(u,u,v) + b(v,p) + b(u,q)$

nonlinear problem:

Find $(u,p) \in V(g) \times Q$ s.t. $A(u,p\,;\,v,q) = (f,v)\ \forall (v,q) \in V \times Q$.

$a_1(\cdot,\cdot,\cdot)$: trilinear form,

$$\begin{aligned} a_1(u+\delta u, u+\delta u, v) &= a_1(u, u+\delta u, v) + a_1(\delta u, u+\delta u, v) \\ &= a_1(u,u,v) + a_1(u,\delta u,v) + a_1(\delta u,u,v) + a_1(\delta u,\delta u,v) \end{aligned}$$

$$\begin{aligned} &A(u+\delta u, p+\delta p\,;\,v,q) - A(u,p\,;\,v,q) \\ &= a(u+\delta u,v) - a(u,v) \\ &\quad + b(v,p+\delta p) - b(v,p) + b(u+\delta u,q) - b(u,q) \\ &\quad + a_1(u+\delta u,u+\delta u,v) - a_1(u,u,v) \\ &= a(\delta u,v) + b(v,\delta p) + b(\delta u,q) + a_1(\delta u,u,v) + a_1(u,\delta u,v) + O(||\delta u||^2) \end{aligned}$$

Find $(\delta u,\delta p) \in V \times Q$ s.t.

$$\begin{aligned} a(\delta u,v) + b(v,\delta p) + b(\delta u,q) + a_1(\delta u,u,v) + a_1(u,\delta u,v) = \\ - A(u,p\,;\,v,q) \quad \forall (v,q) \in V \times Q \end{aligned}$$

Newton iteration

$(u_0, p_0) \in V(g) \times Q$
loop $n = 0, 1 \dots$
  Find $(\delta u, \delta p) \in V \times Q$ s.t.

$$a(\delta u, v) + b(v, \delta p) + b(\delta u, q) + a_1(\delta u, u_n, v) + a_1(u_n, \delta u, v) = A(u_n, p_n\,;\, v, q) \quad \forall (v, q) \in V \times Q$$

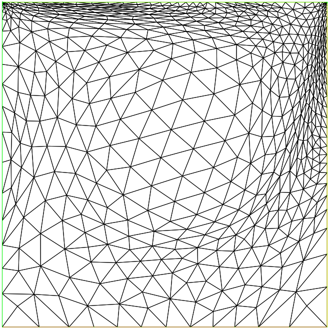
  if $||(\delta u, \delta p)||_{V\times Q} \leq \varepsilon$ then break
  $u_{n+1} = u_n - \delta u$,
  $p_{n+1} = p_n - \delta p$.
loop end.

▸ example4.edp

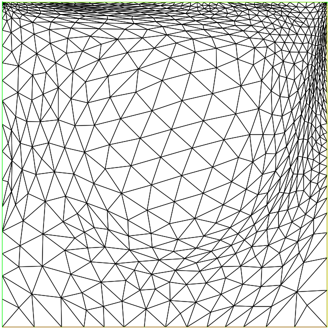
$(u^{(0)}, p^{(0)}) \in V(g) \times Q$: solution of the Stokes eqs., $\nu = 1$.
while ($\nu > \nu_{\mathrm{min}}$)
  Newton iteration $(u^{(k+1)}, p^{(k+1)}) \in V(g) \times Q$ from $(u^{(k)}, p^{(k)})$.
  $\nu = \nu/2$, k++.
while end.
initial guess from the stationary state of lower Reynolds number

mesh adaptation

```
fespace XXMh(Th,[P2,P2,P1]);
XXMh [u1,u2,p];
real lerr=0.01;
Th=adaptmesh(Th, [u1,u2], p, err=lerr, nbvx=100000);
[u1,u2,p]=[u1,u2,p]; // interpolation on the new mesh
```



$\Rightarrow$



`err` : P_1 interpolation error level

`nbvx` : maximum number of vertexes to be generated.

FreeFem++ for non-linear problem

- Jacobian of Newton method needs to be obtained from differential calculus
- good initial guess is necessary to achieve fast convergence of Newton method
- combination with mesh adaptation is an effective technique to improve initial conditions for Newton method

Outline

- Weak formulation for partial differential equations with 2nd order derivatives
- Finite element stiffness matrix from weak form
- Nonlinear finite element solution by Newton method
- Syntax and data structure of FreeFem++ language
  - control flow
  - data structure; array and FE object
  - sparse matrix
  - function macro
  - procedure to compute SpMV for built-in iterative solver
- Schwarz algorithm as preconditioner for global Krylov iteration
- Time-dependent Navier-Stokes equations around a cylinder

syntax of FreeFem++ script

for loop

```
for (int i=0; i<10; i++) {
  ...
  if (err < 1.0e-6) break;
}
```

while loop

```
int i = 0;
while (i < 10) {
  ...
  if (err < 1.0e-6) break;
  i++;
}
```

procedure (function)

```
func real[int] ff(real[int] &pp) { // C++ reference
   real [int] qq;
   qq = ... pp ;             // calculate qq from pp
   return qq;                // return data real[int]
}
```

array and finite element object

```
mesh Th=square(20,20);
fespace Xh(Th, P1); // P1 finite element on Th
Xh uh;              // P1 finite element object
int n = Xh.ndof;    // DOF of FE space Xh
real[int] u(n);     //
u = uh[];           // copy data from FE object to array
for (int i = 0; i < n; i++)
  u(i)=uh[](i);     // index-wise copy, but slow

real[int] u;      // not yet allocated
u.resize(10);     // same as C++ STL vector
real[int] vv = v; // allocated as same size of v.n
a(2)=0.0  ;       // set value of 3rd index
a += b;           // a(i) = a(i) + b(i)
a = b .* c  ;      // a(i) = b(i) * c(i); element-wise
a = b < c ? b : c // a(i) = min(b(i), c(i)); C-syntax
a.sum;            // sum a(i);
a.n;              // size of array
```

There are other operations such as $\ell^1, \ell^2, \ell^\infty$-norms, max, min.
cf. Finite element analysis by mathematical programming language FreeFem++, Ohtsuka-Takaishi [2014].

vector-valued finite element object and 2D array

vector valued FE object

```
mesh Th=square(20,20);
fespace Xh(Th, [P1, P1]); // P1-P1 finite element on Th
Xh [uh1, uh2];            // P1 finite element object

real[int] u(uh1[].n+uh2[].n);   // allocation of array
real[int] v(Xh,nodf);           //
u = [uh1[], uh2[]]; // copy of vector-valued object
v = uh1[];          // u1h[] shows a pointer to an array
                    // in [uh1, uh2]
```

2-D array

```
mesh Th=square(20,20);
fespace Xh(Th, P1); // P1 finite element on Th
Xh[int] uh(10);     // 10 sized array of FE object
real[int,int] uu(Xh.ndof, 10);  // 2D array
for (int m = 0; m < 10; m++) {
  uu(:,m)=uh[m][](:);        // copy each array
}
```

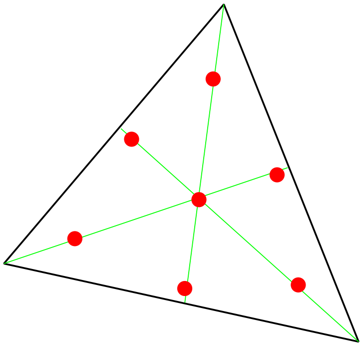
variational form and sparse matrix

```
mesh Th=square(20,20);
fespace Xh(Th, P1);   Xh u, v;
func fnc = sin(pi * x + pi * y);
varf a(u, v) = int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))
             + on(1,u=0.0);
varf f(u, v) = int2d(Th)(fnc * v);
matrix AA = a(Xh, Xh, solver=UMFPACK);
real[int] x(Xh.ndof), b(X.ndof);
b = f(Xh, 0);              // RHS from external force
x = AA^-1 * b;             // solution of linear system

int n = 10;                // example to generate
real[int, int] mm(n, n); // sparse matrix from 2D array
Xh [int] uh(n);
for (int i = 0; i < n; i++)
  uh[i] = sin(pi * real(i+1) * x);
for (int j = 0; j < n; j++)
   for (int i = 0; i < n; i++)
      mm(i, j)=uh[i][]'*uh[j][]; //' inner product
matrix aa = mm;            // copy into sparse matrix
set (aa, solver=UMFPACK);// set sparse solver
```

function macro and FEM object

```
func fnc = sin(pi*x)*sin(pi*y); // function with x,y
mesh Th=square(20,20);
fespace Vh(Th, P2);
Vh fh;
fh = fnc;             // P2 interpolation using mesh data
real sum0 = int2d(Th)(fh * fh);  //numerical quadrature
                                 // applied to P2 FE object
real sum1 = int2d(Th)(fnc * fnc);//numerical quadrature
                                 // applied to (x,y)-func.
```



quadrature `qf5pT`

procedure and CG built-in function : 1/2

▸ example5.edp ▸▸ varf+matrix

```
int n = 20;
mesh Th=square(n,n);
fespace Vh(Th,P1); Vh u,v;
func f = 5.0/4.0*pi*pi*sin(pi*x)*sin(pi*y/2.0);
func h = (-pi)/2.0*sin(pi*x);
func g = sin(pi*x)*sin(pi*y/2.0);
varf aa(u,v) = int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))
               + on(2,3,4,u=1.0);
varf external(u,v) = int2d(Th)(f*v) + int1d(Th,1)(h*v);
real tgv=1.0e+30;
matrix A;
real[int] bc = aa(0, Vh, tgv=tgv);
func real[int] opA(real[int] &pp)
{
  pp = bc ? 0.0 : pp;    // SpMV operation only for inner
  real[int] qq = A * pp;// node without Dirichlet bdry.
  pp = bc ? 0.0 : qq;
  return pp;
}
```

procedure and CG built-in function : 2/2

```
func real[int] opQ(real[int] &pp) // diagonal scaling
{
  for (int i = 0; i < pp.n; i++) {
    pp(i) = pp(i) / A(i, i);
  }
  pp = bc ? 0.0 : pp;
  return pp;
}

A = aa(Vh, Vh, tgv=tgv, solver=sparsesolver);
real[int] ff = external(0, Vh);
v = g;                    // g is valid only on the bdry 1
u[] = bc ? v[] : 0.0; // u[] has Dirichlet data w/o tgv
v[] = A * u[];            // v[] = A_{12}*g
v[] = bc ? 0.0 : v[];
ff -= v[];                // ff_{1} -= A_{12}*u_{2}
ff = bc ? 0.0 : ff;
LinearCG(opA, u[], ff, precon=opQ, nbiter=200,
         eps=1.0e-10,verbosity=50);
```

Outline

Weak formulation for partial differential equations with 2nd order derivatives

Finite element stiffness matrix from weak form

Nonlinear finite element solution by Newton method

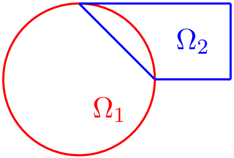
Syntax and data structure of FreeFem++ language

Schwarz algorithm as preconditioner for global Krylov iteration
- overlapping subdomains and RAS/ASM
- 2-level algorithm with a coarse space

Time-dependent Navier-Stokes equations around a cylinder

Thermal convection problem by Rayleigh-Bénard eqs.

Schwarz algorithm : 1/2



Jacobi-Schwarz algorithm

u_1^0, u_2^0 : given

loop $n = 0, 1, 2, \cdots$

$$-\triangle u_1^{n+1} = f \text{ in } \Omega_1 \qquad -\triangle u_2^{n+1} = f \text{ in } \Omega_2$$

$$u_1^{n+1} = 0 \text{ on } \partial\Omega_1 \cap \partial\Omega \qquad u_2^{n+1} = 0 \text{ on } \partial\Omega_2 \cap \partial\Omega$$

$$u_1^{n+1} = u_2^n \text{ on } \partial\Omega_1 \cap \bar{\Omega}_2 \qquad u_2^{n+1} = u_1^n \text{ on } \partial\Omega_2 \cap \bar{\Omega}_1$$

- parallel computation, extendable to more than two subdomains
- overlapping subdomain

Restricted Additive Schwarz algorithm : 1/2

partition of unity

$u = \sum_{i=1}^{2} E_i(\chi_i u_i)$

- E_i : extension from Ω_i to Ω
- χ_i : partition of unity function in $\bar{\Omega}_i$

loop $n = 0, 1, 2, \cdots$

$$-\triangle w_i^{n+1} = f \text{ in } \Omega_i$$
$$w_i^{n+1} = 0 \text{ on } \partial\Omega_i \cap \partial\Omega$$
$$w_i^{n+1} = u^n \text{ on } \partial\Omega_i \cap \bar{\Omega}_j$$
$$u^{n+1} = \sum_{i=1}^{2} E_i(\chi_i w_i^{n+1})$$

by substituting $\triangle u^n$ that satisfies $u^n = 0$ on $\partial\Omega_i \cap \partial\Omega$

$$-\triangle w_i^{n+1} + \triangle u^n = f + \triangle u^n \text{ in } \Omega_i$$
$$w_i^{n+1} - u^n = 0 \text{ on } \partial\Omega_i \cap \partial\Omega, \qquad w_i^{n+1} - u^n = 0 \text{ on } \partial\Omega_i \cap \bar{\Omega}_j$$
$$u^{n+1} - u^n = \sum_{i=1}^{2} E_i(\chi_i w_i^{n+1}) - \sum_{i=1}^{2} E_i(\chi_i u_i^n)$$
$$= \sum_{i=1}^{2} E_i(\chi_i (w_i^{n+1} - u^n))$$

Restricted Additive Schwarz algorithm : 2/2

Restricted additive Schwarz (RAS) algorithm

u_1^0, u_2^0 : given

loop $n = 0, 1, 2, \cdots$

$$r^n = f + \triangle u^n$$

$$-\triangle v_i^{n+1} = r^n \text{ in } \Omega_i$$

$$v_i^{n+1} = 0 \text{ on } \partial\Omega_i$$

$$u^{n+1} = u^n + \sum_{i=1}^{2} E_i(\chi_i v_i^{n+1})$$

RAS $\Rightarrow$ Jacobi-Schwarz method

proof by induction

$$-\triangle(u^n + v_i^n) = -\triangle(u^n) + r^n = f \text{ in } \Omega_i$$

$$u^n + v_i^n = u^n \text{ on } \partial\Omega_i \cap \partial\Omega\,.$$

$$u^n = E_1(\chi_1 u_1^n) + E_2(\chi_2 u_2^n)$$

$$u^n = E_1(0 \cdot u_1^n) + E_2(1 \cdot u_2^n) = u_2^n \text{ on } \partial\Omega_i \cap \bar{\Omega}_2\,.$$

Additive Schwarz algorithm

loop $n = 0, 1, 2, \cdots$

$$-\triangle w_i^{n+1} = f \text{ in } \Omega_i$$

$$w_i^{n+1} = 0 \text{ on } \partial\Omega_i \cap \partial\Omega$$

$$w_i^{n+1} = u^n \text{ on } \partial\Omega_i \cap \bar{\Omega}_j$$

$$u^{n+1} = \textstyle\sum_{i=1}^{2} E_i(w_i^{n+1})$$

Additive Schwarz Method (ASM)

u_1^0, u_2^0 : given

loop $n = 0, 1, 2, \cdots$

$$r^n = f + \triangle u^n$$

$$-\triangle v_i^{n+1} = r^n \text{ in } \Omega_i$$

$$v_i^{n+1} = 0 \text{ on } \partial\Omega_i$$

$$u^{n+1} = u^n + \sum_{i=1}^{2} E_i(v_i^{n+1})$$

Schwarz methods as preconditioner

ASM preconditioner

$$M_{\text{ASM}}^{-1} = \sum_{p=1}^{M} R_p^T (R_p A R_p^T)^{-1} R_p$$

ASM does not converge as fixed point iteration, but M_{ASM}^{-1} is symmetric and works well as a preconditioner for CG method.

RAS preconditioner

$$M_{\text{RAS}}^{-1} = \sum_{p=1}^{M} R_p^T D_p (R_p A R_p^T)^{-1} R_p$$

RAS does converge but M_{RAS}^{-1} is not symmetric and then works as a preconditioner for GMRES method.
convergence of ASM, RAS: slow for many subdomains

$\Rightarrow$ coarse space

2-level Schwarz methods with a coarse space

coarse space by Nicolaides

D_p : discrete representation of the partition of unity

$$\sum_{p=1}^{M} R_p^T D_p R_p = I_N,$$

$\{\vec{z}_p\} \subset \mathbb{R}^N$: basis of coarse space, $Z = [\vec{z}_1, \cdots, \vec{z}_M]$.

$$\vec{z}_p = R_p^T D_p R_p \vec{1},$$

$R_0 = Z^T$.

2-level ASM preconditioner

$$M_{\text{ASM},2}^{-1} = R_0^T (R_0 A R_0^T)^{-1} R_0 + \sum_{p=1}^{M} R_p^T (R_p A R_p^T)^{-1} R_p$$

2-level RAS preconditioner

$$M_{\text{RAS},2}^{-1} = R_0^T (R_0 A R_0^T)^{-1} R_0 + \sum_{p=1}^{M} R_p^T D_p (R_p A R_p^T)^{-1} R_p$$

FreeFem++ script for a 3D domain computation

Subdomain problems : solved by a direct solver
can be distributed in paralle machine.

▸ example6.edp

- domain decomposition by METIS
  `metisdual()`
- overlapping subdomain from non-overlapping one using numerical diffusion operator
  `func bool AddLayers()`
- partition of unity
  `func bool SubdomainsPartitionUnity()`

cf. V. Dolean, P Jolivet, F. Nataf, An Introduction to Domain Decomposition Methods – Algorithms, Theory, and Parallel Implementation, SIAM, 2015. ISBN 978-1-611974-05-8

Outline

Weak formulation for partial differential equations with 2nd order derivatives

Finite element stiffness matrix from weak form

Nonlinear finite element solution by Newton method

Syntax and data structure of FreeFem++ language

Schwarz algorithm as preconditioner for global Krylov iteration

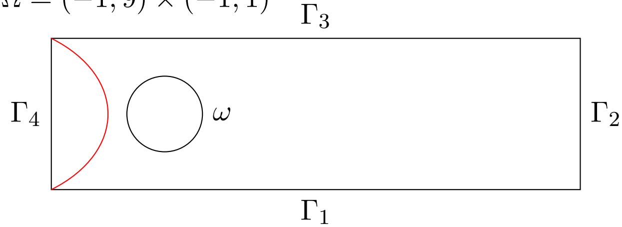
Time-dependent Navier-Stokes equations around a cylinder

- boundary conditions of incompressible flow around a cylinder

Thermal convection problem by Rayleigh-Bénard eqs.

incompressible flow around a cylinder : boundary conditions

$\Omega = (-1, 9) \times (-1, 1)$



$$\frac{\partial u}{\partial t} + u \cdot \nabla u - 2\nu \nabla \cdot D(u) + \nabla p = 0 \text{ in } \Omega$$

$$\nabla \cdot u = 0 \text{ in } \Omega$$

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

boundary conditions:

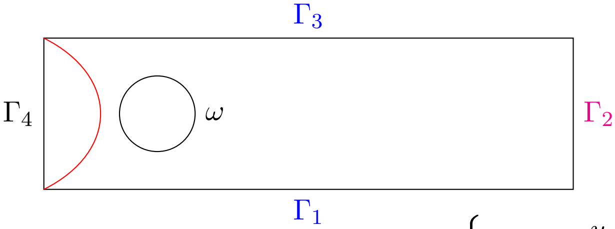
Poiseuille flow on Γ_4 : $u = (1 - y^2, 0)$.

slip boundary condition on $\Gamma_1 \cup \Gamma_3$: $\begin{cases} u \cdot n = 0 \\ (2\nu D(u)n - np) \cdot t = 0 \end{cases}$

no-slip boundary condition on ω : $u = 0$

outflow boundary condition on Γ_2 : $2\nu D(u)n - np = 0$

slip boundary conditions and function space



slip boundary condition on $\Gamma_1 \cup \Gamma_3$: $\begin{cases} u \cdot n = 0 \\ (2\nu\, D(u)n - n\, p) \cdot t = 0 \end{cases}$

- $V(g) = \{v \in H^1(\Omega)^2\,;\, v = g$ on $\Gamma_4 \cup \omega,\ v \cdot n = 0$ on $\Gamma_1 \cup \Gamma_3\}$,
- $Q = L^2(\Omega)$.

non-slip

$$\begin{aligned}\int_{\Gamma_1\cup\Gamma_3} (2\nu D(u)n - n\,p) \cdot v ds &= \int_{\Gamma_1\cup\Gamma_3} (2\nu D(u)n - n\,p) \cdot (v_n n + v_t t) ds \\ &= \int_{\Gamma_1\cup\Gamma_3} (2\nu D(u)n - n\,p) \cdot (v \cdot n) n\, ds \\ &\quad + \int_{\Gamma_1\cup\Gamma_3} (2\nu D(u)n - n\,p) \cdot t v_t\, ds = 0\end{aligned}$$

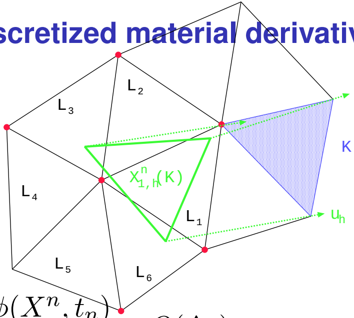
characteristic line and material derivative

$u(x_1, x_2, t) : \Omega \times (0, T] \to \mathbb{R}^2$, given velocity field.
$\phi(x_1, x_2, t) : \Omega \times (0, T] \to \mathbb{R}$.
$X(t) : (0, T] \to \mathbb{R}^2$, characteristic curve :
$\dfrac{dX}{dt}(t) = u(X(t), t), X(0) = X_0$

$$\begin{aligned}\frac{d}{dt}\phi(X(t), t) &= \nabla\phi(X(t), t) \cdot \frac{d}{dt}X(t) + \frac{\partial}{\partial t}\phi(X(t), t) \\ &= \nabla\phi(X(t), t) \cdot u(X(t), t) + \frac{\partial}{\partial t}\phi(X(t), t)\end{aligned}$$

material derivative : $\dfrac{D\phi}{Dt} = \dfrac{\partial}{\partial t}\phi + u \cdot \nabla\phi$.

approximation by difference
$$\frac{D\phi(X(t), t)}{Dt} \sim \frac{\phi(X(t), t) - \phi(X(t - \Delta t), t - \Delta t)}{\Delta t}$$



approximation by Euler method :

$t_n < t_{n+1}$, $t_{n+1} = \Delta t + t_n$.

$$X(t_{n+1}) = x$$

$$X(t_n) = X^n(x) + O(\Delta t^2)$$

$$X^n(x) = x - u(x, t_n)$$

$$\frac{D\phi(X(t_{n+1}), t_{n+1})}{Dt} = \frac{\phi(x, t_{n+1}) - \phi(X^n, t_n)}{\Delta t} + O(\Delta t)$$

$$\sim \frac{\phi^{n+1} - \phi^n \circ X^n}{\Delta t}.$$

u^n : obtained in the previous time step.

Find $(u^{n+1}, p^{n+1}) \in V(g) \times Q$ s.t.

$$\left(\frac{u^{n+1} - u^n \circ X^n}{\Delta t}, v\right) + a(u^{n+1}, v) + b(v, p^{n+1}) = 0 \quad \forall v \in V,$$

$$b(u^{n+1}, q) = 0 \quad \forall q \in Q.$$

FreeFem++ script using characteristic Galerkin method

FreeFem++ provides convect to compute $(u^n \circ X^n, \cdot)$.

example7.edp

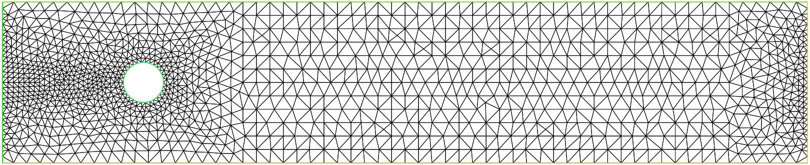
```
real nu=1.0/Re;
real alpha=1.0/dt;
int i;
problem NS([u1,u2,p],[v1,v2,q],solver=UMFPACK,init=i) =
  int2d(Th)(alpha*(u1*v1 + u2*v2)
       +2.0*nu*(dx(u1)*dx(v1)+2.0*d12(u1,u2)*d12(v1,v2)
               +dy(u2)*dy(v2))
       - p * div(v1, v2) - q * div(u1, u2))
- int2d(Th)(alpha*( convect([up1,up2],-dt,up1)*v1
                   +convect([up1,up2],-dt,up2)*v2) )
+ on(1,3,u2=0)+on(4,u1=1.0-y*y,u2=0)+on(5,u1=0,u2=0);

for (i = 0; i <= timestepmax; i++) {
   up1 = u1; up2 = u2; pp = p;
   NS;            // factorization is called when i=0
   plot([up1,up2],pp,wait=0,value=true,coef=0.1);
}
```

FreeFem++ script for mesh generation around a cylinder

Delaunay triangulation from nodes given on the boundary
boundary segments are oriented and should be connected.

```
int n1 = 30;
int n2 = 60;
border ba(t=0,1.0){x=t*10.0-1.0;y=-1.0;label=1;};
border bb(t=0,1.0){x=9.0;y=2.0*t-1.0;label=2;};
border bc(t=0,1.0){x=9.0-10.0*t;y=1.0;label=3;};
border bd(t=0,1.0){x=-1.0;y=1.0-2.0*t;label=4;};
border cc(t=0,2*pi){x=cos(t)*0.25+0.75;
                    y=sin(t)*0.25;label=5;};
mesh Th=buildmesh(ba(n2)+bb(n1)+bc(n2)+bd(n1)+cc(-n1));
plot(Th);
```



stream line for visualization of flow around a cylinder : 1/2

stream function $\psi : \Omega \to \mathbb{R}$, $u = \begin{bmatrix} \partial_2 \psi \\ -\partial_1 \psi \end{bmatrix}$.

boundary conditions for the stream line:

inlet: $y - \dfrac{y^3}{3} = \displaystyle\int_0^y u_1(x_1, t) dt = \int_0^y \partial_2 \psi(x_1, t) dt = \psi(x_1, y) - \psi(x_1, 0)$

$$\psi(x_1, y) = \psi(x_1, 0) + y - \frac{y^3}{3} = y - \frac{y^3}{3}.$$

slip: $0 = \displaystyle\int_{x_1}^x u_2(t, \pm 1) dt = \int_{x_1}^x -\partial_1 \psi(t, \pm 1) dt = \psi(x_1, \pm 1) - \psi(x, \pm 1)$

$$\psi(x, \pm 1) = \psi(x_1, \pm 1) = \psi(x_1, 0) \pm 2/3 = \pm 2/3.$$

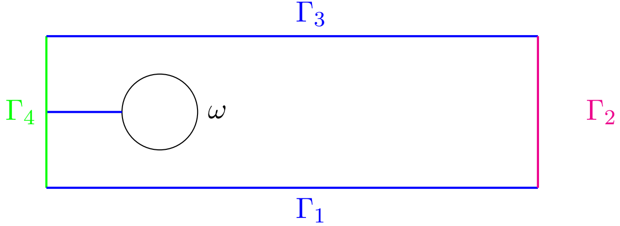
cylinder: $0 = \displaystyle\int_{-\pi}^{\theta} u \cdot n \, d\theta = \int_{-\pi}^{\theta} -\partial_1 \psi \, r \sin\theta + \partial_2 \psi \, r \cos\theta$

$$= \int_{-\pi}^{\theta} \frac{\partial}{\partial \theta} \psi(r, \theta) d\theta.$$

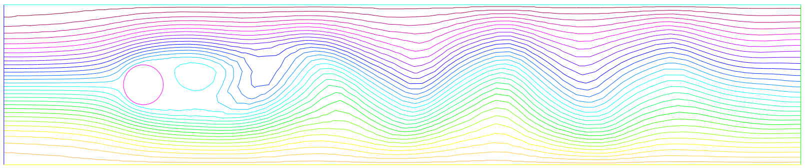
center from inlet: $u_2 = 0 \ \Rightarrow$ same as slip wall ,

$$\psi|_{\omega} = \psi(x_1, 0) = 0.$$

stream line for visualization of flow around a cylinder : 2/2



slip boundary condition on $\Gamma_1 \cup \Gamma_3$, outflow on Γ_2.



Outline

Weak formulation for partial differential equations with 2nd order derivatives

Finite element stiffness matrix from weak form

Nonlinear finite element solution by Newton method

Syntax and data structure of FreeFem++ language

Schwarz algorithm as preconditioner for global Krylov iteration

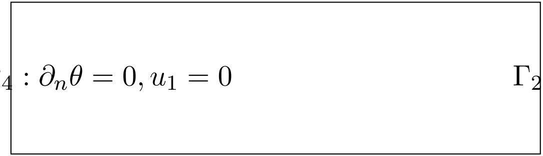
Time-dependent Navier-Stokes equations around a cylinder

Thermal convection problem by Rayleigh-Bénard eqs.

- governing equations by Boussinesq approximation
- stationary solution by Newton method from numerical stationary solution

thermal convection in a box : 1/2

$\Gamma_3 : \theta = \theta_0, u_2 = 0$



$\Gamma_4 : \partial_n \theta = 0, u_1 = 0$ $\quad$ $\Gamma_2 : \partial_n \theta = 0, u_1 = 0$

$\Gamma_1 : \theta = \theta_0 + \Delta\theta, u_2 = 0$

Rayleigh-Bénard equations

$$\rho \left(\frac{\partial u}{\partial t} + u \cdot \nabla u \right) - 2\nabla \cdot \mu_0 D(u) + \nabla p = -\rho\, g \vec{e}_2 \text{ in } \Omega\,,$$

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

$$\frac{\partial \theta}{\partial t} + u \cdot \nabla \theta - \nabla \cdot (\kappa \theta) = 0 \text{ in } \Omega\,.$$

$\vec{e}_2$: unit normal of y-direction
d : height of the box, g : gravity acceleration,
κ : thermal diffusivity, μ_0 : viscosity

thermal convection in a box : 2/2

Boussinesq approximation : $\rho = \rho_0\{1 - \alpha(\theta - \theta_0)\}$, $\theta_0 = 0$.
ρ_0: representative density, α : thermal expansion coefficient.
non-dimensional Rayleigh-Bénard equations

$$\frac{1}{Pr}\left(\frac{\partial u}{\partial t} + u \cdot \nabla u\right) - 2\nabla \cdot D(u) + \nabla p = Ra\theta\vec{e}_2 \text{ in } \Omega\,,$$

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

$$\frac{\partial \theta}{\partial t} + u \cdot \nabla\theta - \triangle\theta = 0 \text{ in } \Omega$$

$$u \cdot n = 0 \text{ on } \partial\Omega\,,$$

$$\theta = 1 \text{ on } \Gamma_1\,,$$

$$\theta = 0 \text{ on } \Gamma_3\,,$$

$$\partial_n\theta = 0 \text{ on } \Gamma_2 \cup \Gamma_4\,.$$

- $Pr = \dfrac{\mu_0}{\kappa\rho_0}$: Prandtl number,
- $Ra = \dfrac{\rho_0 g\alpha\Delta\theta d^3}{\kappa\mu_0}$: Rayleigh number.

a weak form to solve time-dependent Rayleigh-Bénard eqs.

- velocity : $V = \{v \in H^1(\Omega)^2\,;\, v\cdot n = 0$ on $\partial\Omega\}$,
- pressure : $Q = L^2_0(\Omega) = \{p \in L^2(\Omega)\,;\, \int_\Omega p\,dx = 0\}$,
- temperature : $\Psi_D = \{\theta \in H^1(\Omega)\,;\, \theta = 1$ on $\Gamma_1, \theta = 0$ on $\Gamma_3\}$.

bilinear forms:

$a_0(u,v) = \int_\Omega 2D(u) : D(v), \quad b(v,p) = -\int_\Omega \nabla\cdot v\, p,$

$c_0(\theta,\psi) = \int_\Omega \nabla\theta\cdot\nabla\psi$.

using Characteristic Galerkin method: example8.edp

$(u^n,\theta^n) \in V \times \Psi_D$: from previous time step

Find $(u^{n+1}, p^{n+1}, \theta^{n+1}) \in V \times Q \times \Psi_D$ s.t.

$$\frac{1}{Pr}\left(\frac{u^{n+1} - u^n\circ X^n}{\Delta t}, v\right) + a_0(u^{n+1}, v) + b(v, p^{n+1}) = Ra(\theta^n \vec{e}_2, v) \quad \forall v \in V,$$

$$b(u^{n+1}, q) = 0 \quad \forall q \in Q,$$

$$\left(\frac{\theta^{n+1} - \theta^n\circ X^n}{\Delta t}, \psi\right) + c_0(\theta^{n+1}, \psi) = 0 \quad \forall \psi \in \Psi_0.$$

a weak form to solve stationary Rayleigh-Bénard eqs.

trilinear forms and bilinear form for the Navier-Stokes eqs.

- $a_1(u, v, w) = \frac{1}{Pr}\int_\Omega (u \cdot \nabla v) \cdot w$
- $c_1(u, \theta, \psi) = \int_\Omega (u \cdot \nabla \theta) \cdot \psi$
- $A(u, p\,;\, v, q) = a_0(u, v) + a_1(u, u, v) + b(v, p) + b(u, q)$

Newton iteration $(u_0, p_0, \theta_0) \in V \times Q \times \Psi_D$ ▸ example9.edp

loop $n = 0, 1 \ldots$

Find $(\delta u, \delta p, \delta\theta) \in V \times Q \times \Psi_0$ s.t.

$$a_0(\delta u, v) + b(v, \delta p) + b(\delta u, q) + a_1(\delta u, u_n, v) + a_1(u_n, \delta u, v)$$
$$-Ra(\delta\theta\vec{e}_2, v) = A(u_n, p_n\,;\, v, q) - Ra(\theta_n\vec{e}_2, v) \quad \forall (v, q) \in V \times Q$$
$$c_0(\delta\theta, \psi) + c_1(u_n, \delta\theta, \psi) + c_1(\delta u, \theta_n, \psi) = c_0(\theta_n, \psi) + c_1(u_n, \theta_n, \psi)$$
$$\forall \psi \in \Psi_0$$

if $||(\delta u, \delta p, \delta\theta)||_{V\times Q\times\Psi} \leq \varepsilon$ then break

$u_{n+1} = u_n - \delta u, \quad p_{n+1} = p_n - \delta p, \quad \theta_{n+1} = \theta_n - \delta\theta$.

loop end.

initial data $\Leftarrow$ stationary solution by time-dependent problem.

Application of finite element method to fluid problems

Time-dependent Navier-Stokes equations

- material derivative is approximated by Characteristic Galerkin method
- functional space of pressure depends on boundary conditions of flow, e.g., inflow, non-slip, slip, and outflow.

Thermal convection problem by Rayleigh-Bénard equations

- time-dependent problems for fluid and temperature by convection are solved by Characteristic Galerkin method
- stationary solution is obtained by Newton iteration using an initial value obtained from time-evolutionary solution

References

FreeFem++:

- F. Hecht, FreeFem++ manual, 3rd ed., 2015.
- K. Ohtsuka, T. Takaishi, Finite element analysis by mathematical programming language FreeFem++ (in Japanese), Industrial and Applied Mathematics Series Vol.4, Kyoritsu, 2014.
- I. Danaila, F. Hecht, O. Pironneau, Simulation numérique en C++, Dunod, 2003.

Finite element theory:

- D. Braess, Finite elements – Theory, fast solvers and application in solid mechanics, 3rd ed., Cambridge Univ. Press, 2007.
- A. Ern, J.-L. Guermond, Theory and practice of finite elements, Springer Verlag, New-York, 2004.
- M. Tabata, Numerical solution of partial differential equations II (in Japanese), Iwanami Shoten, 1994.

```
// example 1 : poisson-mixedBC.edp   [slide page 8]
// finite element solution of Poisson equation with mixed boundary condition
// for RIIT Tutorial at Kyushu University, 25 Nov.2016, Atsushi Suzuki

int n = 20;
mesh Th=square(n,n);
fespace Vh(Th,P1);

Vh uh,vh;
real err, hh;

func f = 5.0/4.0 * pi * pi * sin(pi * x) * sin(pi * y / 2.0);
func h = (-pi)/2.0 * sin(pi * x);
func g = sin(pi * x) * sin(pi * y / 2.0);
// for error estimation
func sol = sin(pi * x) * sin(pi * y / 2.0);
func solx = pi * cos(pi * x) * sin(pi * y / 2.0);
func soly = (pi / 2.0) * sin(pi * x) * cos(pi * y / 2.0);

solve poisson(uh,vh) =
int2d(Th)( dx(uh)*dx(vh)+dy(uh)*dy(vh) )
-  int2d(Th)( f*vh )
-  int1d(Th,1) (h * vh)
+  on(2,3,4,uh=g);

hh = 1.0 / real(n) * sqrt(2.0);

// int2d uses qf5pT : 5th order integration quadrature
err = int2d(Th)( (dx(uh) - solx) * (dx(uh) - solx) +
                 (dy(uh) - soly) * (dy(uh) - soly) +
                 (uh - sol) * (uh - sol));
err = sqrt(err);

cout << "DOF=" << uh[].n << "\t h=" << hh << " err-H1=" << err << endl;
plot(uh,wait=1);
```

```
// example 2 : stokes-mixedBC.edp   [slide page 12]
// error estimation for finite element solution of Stokes equations
// with P2/P1 or P1b/P1 element
// for RITT Tutorial at Kyushu University, 25 Nov.2016, Atsushi Suzuki

int n1 = 20;
int n2 = n1 * 2;
mesh Th1=square(n1,n1);
mesh Th2=square(n2,n2);

// finite element spaces
fespace Vh1(Th1,P2),Qh1(Th1,P1);
fespace Vh2(Th2,P2),Qh2(Th2,P1);

// fespace Vh1(Th1,P1),Qh1(Th1,P1);
// fespace Vh2(Th2,P1),Qh2(Th2,P1);

// fespace Vh1(Th1,P2),Qh1(Th1,P0);
// fespace Vh2(Th2,P2),Qh2(Th2,P0);

// external force
func f1 = 5.0/8.0 * pi * pi * sin(pi * x) * sin(pi * y / 2.0) + 2.0 * x;
func f2 = 5.0/4.0 * pi * pi * cos(pi * x) * cos(pi * y / 2.0) + 2.0 * y;
func h1 = 3.0/4.0 * pi * sin(pi * x) * cos(pi * y / 2.0);
func h2 = pi * cos(pi * x) * sin(pi * y / 2.0) + x * x + y * y;
func sol1 = sin(pi * x) * sin(pi * y / 2.0) / 2.0;
func sol2 = cos(pi * x) * cos(pi * y / 2.0);
func solp = x * x + y * y;

func sol1x = pi * cos(pi * x) * sin(pi * y / 2.0) / 2.0;
func sol1y = pi / 2.0 * sin(pi * x) * cos(pi * y / 2.0) / 2.0;

func sol2x = (-pi) * sin(pi * x) * cos(pi * y / 2.0);
func sol2y = (-pi / 2.0) * cos(pi * x) * sin(pi * y / 2.0);

// finite element solutions and test functions
Vh1 u11,u12,v11,v12;
Qh1 p1,q1;
Vh2 u21,u22,v21,v22;
Qh2 p2,q2;

real epsln = 1.0e-6;
// definitions of macros for strain rate tensor
macro d11(u1)     dx(u1) //
macro d22(u2)     dy(u2) //
macro d12(u1,u2)  (dy(u1) + dx(u2))/2.0 //

// stokes problem
solve stokes1(u11,u12,p1, v11,v12,q1) =
  int2d(Th1)(
        2.0*(d11(u11)*d11(v11)+2.0*d12(u11,u12)*d12(v11,v12)+d22(u12)*d22(v12))
        - p1*dx(v11) - p1*dy(v12)
        - dx(u11)*q1 - dy(u12)*q1)
- int2d(Th1)(f1*v11+f2*v12) - int1d(Th1,1)(h1*v11+h2*v12)
+ on(2,3,4,u11=sol1,u12=sol2);

real meanp,err1,err2,hh1,hh2;


plot([u11,u12],p1,wait=1,value=true,coef=0.1);

solve stokes2(u21,u22,p2, v21,v22,q2) =
  int2d(Th2)(
        2.0*(d11(u21)*d11(v21)+2.0*d12(u21,u22)*d12(v21,v22)+d22(u22)*d22(v22))
        - p2*dx(v21) - p2*dy(v22)
        - dx(u21)*q2 - dy(u22)*q2)
- int2d(Th2)(f1*v21+f2*v22) - int1d(Th2,1)(h1*v21+h2*v22)
+ on(2,3,4,u21=sol1,u22=sol2);

plot([u21,u22],p2,wait=1,value=true,coef=0.1);
```

```
hh1 = 1.0 / n1 * sqrt(2.0);
hh2 = 1.0 / n2 * sqrt(2.0);

err1 = int2d(Th1)( (dx(u11) - sol1x) * (dx(u11) - sol1x)
                   + (dy(u11) - sol1y) * (dy(u11) - sol1y)
                   + (u11 - sol1) * (u11 - sol1) +
                   (dx(u12) - sol2x) * (dx(u12) - sol2x)
                   + (dy(u12) - sol2y) * (dy(u12) - sol2y)
                   + (u12 - sol2) * (u12 - sol2) +
                   (p1 - solp) * (p1 - solp));
err1 = sqrt(err1);

err2 = int2d(Th2)( (dx(u21) - sol1x) * (dx(u21) - sol1x)
                   + (dy(u21) - sol1y) * (dy(u21) - sol1y)
                   + (u21 - sol1) * (u21 - sol1) +
                   (dx(u22) - sol2x) * (dx(u22) - sol2x)
                   + (dy(u22) - sol2y) * (dy(u22) - sol2y)
                   + (u22 - sol2) * (u22 - sol2) +
                   (p2 - solp) * (p2 - solp));

err2 = sqrt(err2);


cout << "coarse mesh: h=" << hh1 << " err-H1/L2=" << err1 << endl;
cout << "fine mesh:   h=" << hh2 << " err-H1/L2=" << err2 << endl;
cout << "O(h)=" << log(err1/err2)/log(hh1/hh2) << endl;
```

```
// example 3 : poisson-matrix.edp   [slide page 26]
// finite element solution of Poisson equation with mixed boundary condition
// with P1 element
// for RIIT Tutorial at Kyushu University, 25 Nov.2016, Atsushi Suzuki

int n = 20;
mesh Th=square(n,n);
fespace Vh(Th,P1);

Vh u,v;
real err, hh;

func f = 5.0/4.0 * pi * pi * sin(pi * x) * sin(pi * y / 2.0);
func h = (-pi)/2.0 * sin(pi * x);
func g = sin(pi * x) * sin(pi * y / 2.0);
// for error estimation
func sol = sin(pi * x) * sin(pi * y / 2.0);
func solx = pi * cos(pi * x) * sin(pi * y / 2.0);
func soly = (pi / 2.0) * sin(pi * x) * cos(pi * y / 2.0);

varf aa(u,v) = int2d(Th)( dx(u)*dx(v)+dy(u)*dy(v) )
               + on(2,3,4,u=g); // u=1 is enough to say which is Dirichlet node
varf external(u,v) = int2d(Th)( f*v ) + int1d(Th,1) (h * v)
                     + on(2,3,4,u=g); // inhomogenoues Dirichlet data are given
real tgv=1.0e+30;
matrix A = aa(Vh,Vh,tgv=tgv,solver=CG);
real[int] ff = external(0,Vh,tgv=tgv); // containing Dirichlet data with tgv

u[] = A^-1 * ff;

hh = 1.0 / real(n) * sqrt(2.0);

// int2d uses qf5pT : 5th order integration quadrature
err = int2d(Th)( (dx(u) - solx) * (dx(u) - solx) +
                 (dy(u) - soly) * (dy(u) - soly) +
                 (u - sol) * (u - sol));
err = sqrt(err);

cout << "DOF=" << u[].n << "\t h=" << hh << " err-H1=" << err << endl;
```

```
// example 4 : cavityNewton.edp   [slide page 36]
// stational Navier-Stokes equations in a cavity by Newton iteration
// P2/P1 element
// for RIIT Tutorial at Kyushu University, 25 Nov.2016, Atsushi Suzuki
// based on examples++-tutorial/cavityNewtow.edp

mesh Th=square(40,40);
fespace Xh(Th,P2);
fespace Mh(Th,P1);
fespace XXMh(Th,[P2,P2,P1]);
XXMh [u1,u2,p], [v1,v2,q];

macro d11(u1)      dx(u1) //
macro d22(u2)      dy(u2) //
macro d12(u1,u2)  (dy(u1) + dx(u2))/2.0 //
macro div(u1,u2)  (dx(u1) + dy(u2))//
macro grad(u1,u2) [dx(u1), dy(u2)]//
macro ugrad(u1,u2,v) (u1*dx(v) + u2*dy(v)) //
macro Ugrad(u1,u2,v1,v2) [ugrad(u1,u2,v1), ugrad(u1,u2,v2)]//

real epsln = 1.0e-6;

solve Stokes ([u1,u2,p],[v1,v2,q],solver=UMFPACK) =
  int2d(Th)(2.0*(d11(u1)*d11(v1) + 2.0*d12(u1,u2)*d12(v1,v2) + d22(u2)*d22(v2))
            - p * div(v1,v2) - q * div(u1,u2)
            - p * q * epsln)
  + on(3,u1=4*x*(1-x),u2=0)
  + on(1,2,4,u1=0,u2=0);

plot(coef=0.2,cmm=" [u1,u2] and p  ",p,[u1,u2],wait=1);

Mh psi,phi;

problem streamlines(psi,phi,solver=UMFPACK) =
      int2d(Th)( dx(psi)*dx(phi) + dy(psi)*dy(phi))
   +  int2d(Th)( -phi*(dy(u1)-dx(u2)))
   +  on(1,2,3,4,psi=0);

streamlines;
plot(psi,wait=1);

real nu=1.0;
XXMh [up1,up2,pp];
varf vDNS ([u1,u2,p],[v1,v2,q]) =
  int2d(Th)(nu * 2.0*(d11(u1)*d11(v1)+2.0*d12(u1,u2)*d12(v1,v2)+d22(u2)*d22(v2))
            - p * div(v1, v2) - q * div(u1, u2)
            - p * q * epsln
            + Ugrad(u1,u2,up1,up2)'*[v1,v2] //'
            + Ugrad(up1,up2,u1,u2)'*[v1,v2]) //'
  + on(1,2,3,4,u1=0,u2=0);

// [up1, up2, pp] are obtained from the previous step
varf vNS ([u1,u2,p],[v1,v2,q]) =
  int2d(Th)(nu * 2.0*(d11(up1)*d11(v1)+2.0*d12(up1,up2)*d12(v1,v2)+
                      d22(up2)*d22(v2))
            - pp * div(v1, v2) - q * div(up1, up2)
            - pp * q * epsln
            + Ugrad(up1,up2,up1,up2)'*[v1,v2]) //'
  + on(1,2,3,4,u1=0,u2=0);

Xh uu1=u1, uu2=u2; // initial condition is given by Stokes eqs : Re=0.
up1[] = 0.0; // initialize for [up1, up2, pp]

real reyini = 100.0;
real reymax = 12800.0;
real re = reyini;
int kreymax = log(reymax / reyini)/log(2.0) * 2.0;
for(int k = 0; k < kreymax; k++) {
  re *= sqrt(2.0);
  real lerr=0.02; // parameter to control mesh refinmente
```

```
  if(re>8000) lerr=0.01;
  if(re>10000) lerr=0.005;
  for(int step= 0 ;step < 2; step++) {
    Th=adaptmesh(Th, [u1,u2], p, err=lerr, nbvx=100000);
    [u1, u2, p]=[u1, u2, p];
    [up1, up2, pp]=[up1, up2, pp];
    plot(Th, wait=1);
    for (int i = 0 ; i < 20; i++) {
      nu = 1.0 / re;
      up1[] = u1[];
      real[int] b = vNS(0, XXMh);
      matrix Ans = vDNS(XXMh, XXMh, solver=UMFPACK);
      real[int] w = Ans^-1*b;
      u1[] -= w;
      cout << "iter = "<< i << "  " << w.l2 <<  " Reynolds number = " << re
           << endl;
      if(w.l2 < 1.0e-6) break;
    } // loop : i
  }   // loop : step
  streamlines;
  plot(psi,nbiso=30,wait=1);
  // extract velocity component from [u1, u2, p]
  uu1 = u1;
  uu2 = u2;
  plot(coef=0.2,cmm="rey="+re+" [u1,u2] and p  ",psi,[uu1,uu2],wait=1,nbiso=20);
} // loop : re
```

```
// example 5 : poisson-LinearCG.edp   [slide page 45]
// finite element solution of Poisson equation with mixed boundary condition
// in matrix form and solved by LinearCG with diagonal preconditioner
// for RIIT Tutorial at Kyushu University, 25 Nov.2016, Atsushi Suzuki

int n = 20;
mesh Th=square(n,n);
fespace Vh(Th,P1);

Vh u,v;
real err, hh;

func f = 5.0/4.0 * pi * pi * sin(pi * x) * sin(pi * y / 2.0);
func h = (-pi)/2.0 * sin(pi * x);
func g = sin(pi * x) * sin(pi * y / 2.0);
// for error estimation
func sol = sin(pi * x) * sin(pi * y / 2.0);
func solx = pi * cos(pi * x) * sin(pi * y / 2.0);
func soly = (pi / 2.0) * sin(pi * x) * cos(pi * y / 2.0);

varf aa(u,v) = int2d(Th)( dx(u)*dx(v)+dy(u)*dy(v) )
               + on(2,3,4,u=1.0);
varf external(u,v) = int2d(Th)( f*v ) + int1d(Th,1) (h * v);

real tgv=1.0e+30;
matrix A;
real[int] bc = aa(0, Vh, tgv=tgv);

func real[int] opA(real[int] &pp)
{
  pp = bc ? 0.0 : pp;      // SpMV operation only for node in interior of
  real[int] qq = A*pp;     // the domain without Dirichlet nodes
  pp = bc ? 0.0 : qq;
  return pp;
}

func real[int] opQ(real[int] &pp)
{
  for (int i = 0; i < pp.n; i++) {
    pp(i) = pp(i) / A(i, i);
  }
  pp = bc ? 0.0 : pp;
  return pp;
}

A = aa(Vh, Vh, tgv=tgv, solver=sparsesolver);
real[int] ff = external(0, Vh);
v = g;                // g is valid only on the boundary 1
u[] = bc ? v[] : 0.0; // u[] has Dirichlet data without tgv
v[] = A * u[];        // v[] = A_{12}*g
v[] = bc ? 0.0 : v[];
ff -= v[];            // ff_{1} -= A_{12}*u_{2}
ff = bc ? 0.0 : ff;
LinearCG(opA, u[], ff, precon=opQ, nbiter=200, eps=1.0e-10,verbosity=50);

hh = 1.0 / real(n) * sqrt(2.0);
// int2d uses qf5pT : 5th order integration quadrature
err = int2d(Th)( (dx(u) - solx) * (dx(u) - solx) +
                 (dy(u) - soly) * (dy(u) - soly) +
                 (u - sol) * (u - sol));
err = sqrt(err);

cout << "DOF=" << u[].n << "\t h=" << hh << " err-H1=" << err << endl;
```

Schwarz3d.edp 1/5

```
// example6 : Schwarz3d    [slide page 54]
// Schwarz preconiditoner for Krylov method
// for RIIT Tutorial at Kyushu University, 25 Nov.2016, Atsushi Suzuki
// based on FreeFem++ manual, section 10 and
// scripts in An Introduction to Domain Decomposition Methods --
//    Algorithms, Theory, and Parallel Implementation,
//    V. Dolean, P Jolivet, F. Nataf, SIAM 2015, ISBN 978-1-611974-05-8


load "medit";
load "metis";
include "cube.idp"

bool flagRAS=false;
int sizeoverlaps=1;      // size of overlap

int[int] NN=[50,50,50]; //
bool withmetis=true;
int npart= 8;

// using numerical diffusion of mass matrix : FreeFem++ manual Examples 11.9
func bool AddLayers(mesh3 & Th,real[int] &ssd,int n)
{
  fespace Vh(Th,P1);
  fespace Ph(Th,P0);
  Ph s;
  s[]= ssd;
  Vh u;
  varf vM(u,v)=int3d(Th,qforder=1)(u*v/volume);
  matrix M=vM(Ph,Vh);
  for(int i=0;i<n;++i) {
    u[]= M*s[];
    u = u > 0.1;
    s[]= M'*u[]; //';
    s = s > 0.1;
  }
  ssd=s[];
  return true;
}

func bool SubdomainsPartitionUnity(mesh3 & Th, int nnpart, real[int] & partdof,
                                   int sizeoverlaps,
                                   mesh3[int] & Tha,
                                   matrix[int] & Rih, matrix[int] & Dih)
{
  fespace Vh(Th,P1);
  fespace Ph(Th,P0);

  mesh3 Thi=Th;
  fespace Vhi(Thi,P1);  // FreeFem++ trick, formal definition
  Vhi[int] pun(nnpart), dum(nnpart); // local fem functions Vh sun=0, unssd=s0;
  Vh sun = 0, unssd = 0, demo = 0;
  Ph part;
  part[]=partdof;
  for(int i=0;i<nnpart;++i) {
    Ph suppi= abs(part - i) < 0.1; // boolean 1 in the subdomain 0 elsewhere
    AddLayers(Th,suppi[],sizeoverlaps);     // partitions by adding layers
    Thi=Tha[i]=trunc(Th,suppi>0,label=10,split=1); //  mesh interfaces label 10
    Rih[i]=interpolate(Vhi,Vh,inside=true); // Restriction operator : Vh to Vhi
    pun[i][] = 1.0;
    sun[] += Rih[i]'*pun[i][]; // '
  }
  for(int i=0;i<nnpart;++i) {
    Thi=Tha[i];
    //    medit("Thi"+i,Thi);
    pun[i]= pun[i]/sun;
    Dih[i]=pun[i][]; //diagonal matrix built from a vector if(verbosity > 1)
    dum[i] = (pun[i] == 1.0 ? 0.0 : pun[i]);
    demo[] += Rih[i]'*dum[i][];
  }
```

```
  plot(demo,cmm="overlapped skelton",wait=1);
  return true;
}

real [int,int] BB=[[0,1],[0,1],[0,1]];
int [int,int] L=[[1,1],[1,1],[1,1]];
mesh3 Thg=Cube(NN,BB,L);
//medit("Th",Thg);
fespace Ph(Thg,P0);
fespace Vh(Thg,P1);
//fespace Xh(Thg,[P1,P1,P1]);

Ph  part;
Vh  sun=0,unssd=0;
Ph xx=x,yy=y;
if (withmetis)   {
  int[int] nupart(Thg.nt);
  metisdual(nupart,Thg,npart);
  for(int n=0;n<nupart.n; n++)
    part[][n]=nupart[n];
}
//plot(part,fill=1,cmm="subdomains",wait=false);

mesh3[int] aTh(npart);
matrix[int] Rih(npart);
matrix[int] Dih(npart);
matrix[int] aA(npart);
real[int] partdof(npart);
Vh[int] Z(npart);          // coarse space : only used as set of arrays
matrix E;

SubdomainsPartitionUnity(Thg, npart, part[], sizeoverlaps, aTh, Rih, Dih);

//plot(part,fill=1,cmm="subdomains",wait=1);

macro Grad(u) [dx(u),dy(u),dz(u)] // EOM
func f = 1;      // external force
func g = 0 ;     // homogeneous Dirichlet data
func kappa = 30.; // viscosity
func eta = 0;
Vh rhsglobal,uglob; // rhs and solution of the global problem
varf vaglobal(u,v) = int3d(Thg)((1.0 + (kappa - 1.0) * x * y * z) *
                                Grad(u)'*Grad(v)) //'
                     +on(1,u=1.0);
varf vexternal(u,v)= int3d(Thg)(f*v);
matrix Aglobal;
real tgv=1.e+30;
real[int] bc = vaglobal(0, Vh, tgv=tgv);

Aglobal = vaglobal(Vh,Vh,tgv=tgv,solver = CG); // global matrix
rhsglobal[] = vexternal(0,Vh); // global rhs
rhsglobal[] = bc ? 0.0 : rhsglobal[];
//uglob[] = Aglobal^-1*rhsglobal[];
//plot(uglob,value=1,fill=1,wait=1,cmm="Solution by a direct method",dim=3);

for(int n = 0; n < npart; n++) {
  matrix aT = Aglobal*Rih[n]';  // '
  aA[n] = Rih[n]*aT;
  set(aA[n], solver = sparsesolver);
}

func real[int] opA(real[int] &v)
{
  v = bc ? 0.0 : v;
  real[int] s = Aglobal * v;
  s = bc ? 0.0 : s;
  return s;
}

func real[int] opScale(real[int] &v)
```

```
{
  v = bc ? 0.0 : v;
  real[int] diag(v.n);
  diag = Aglobal.diag;
  real[int] s = v ./ diag;    // division on each element of array
  s = bc ? 0.0 : s;
  return s;
}

func bool CoarseSpace(matrix &EE)
{
  for (int n = 0; n < npart; n++) {
    Z[n] = 1.0;
    real[int] zit = Rih[n] * Z[n][];
    real[int] zitemp = Dih[n] * zit;
    Z[n][] = Rih[n]'*zitemp; //'
  }
  real[int,int] Ef(npart,npart); //
  for(int m = 0; m < npart; m++) {
    real[int] zz = opA(Z[m][]);
    for(int n = 0; n < npart; n++) {
      Ef(m, n) = Z[n][]'*zz; // '
    }
  }
  EE = Ef;
  set(EE,solver=UMFPACK);
  return true;
}

func real[int] opQ(real[int] &v)
{
  real [int] s(v.n);
  s = 0.0;
  real[int] vv(npart);
  v = bc ? 0.0 : v;
  for(int n = 0; n < npart; n++) {
    vv[n]= Z[n][]'*v; // '
  }
  real[int] zz = E^-1 * vv;
  for(int n = 0; n < npart; n++) {
    s +=zz[n] * Z[n][];
  }
  return s;
}

func real[int] opASM(real[int] &v)
{
  real [int] s(v.n);
  s = 0.0;
  v = bc ? 0.0 : v;
  for (int n = 0; n < npart; n++) {
    real[int] bi = Rih[n]*v;
    real[int] ui = aA[n]^-1*bi; // local solve
    if (flagRAS) {
       bi = Dih[n]*ui;
    }
    else {
      bi = ui;               // ASM is appropriate for preconditioner
    }
    s += Rih[n]'*bi; // '
  }
  s = bc ? 0.0 : s;
  return s;
}

func real[int] opASM2(real[int] &v)
{
  real[int] s(v.n);
  s = opQ(v);
  s += opASM(v);
```

```
  return s;
}

func real[int] opP(real[int] &v)  // A-orthogonal projection onto Z^T : I - AQ
{
  real[int] s(v.n);
  real[int] s2(v.n);
  s = opA(v);
  s2 = opQ(s);
  s = v - s2;
  return s;
}

func real[int] opPt(real[int] &v)
{
  real[int] s(v.n);
  real[int] s2(v.n);
  s = opQ(v);
  s2 = opA(s);
  s = v - s2;
  return s;
}

func real[int] opASMQ(real[int] &v)
{
  real[int] s(v.n);
  s = opQ(v);
  real[int] ss(v.n);
  ss = opPt(v);
  real[int] sss(v.n);
  sss = opASM(ss);
  ss = opP(sss);
  s += ss;
  return s;
}

Vh un;

CoarseSpace(E);        // coarse space Vh[int] Z is also generated here
if (flagRAS) {
  un = 0.0;
  LinearGMRES(opA, un[], rhsglobal[], nbiter=400, precon=opScale,
              eps=1.0e-10,verbosity=50);
  un = 0.0;
  LinearGMRES(opA, un[], rhsglobal[], nbiter=200, precon=opASM,
              eps=1.0e-10,verbosity=50);
  un = 0;
  LinearGMRES(opA, un[], rhsglobal[], nbiter=200, precon=opASM2,
              eps=1.0e-10,verbosity=50);
  un= 0;
  LinearGMRES(opA, un[], rhsglobal[], nbiter=200, precon=opASMQ,
              eps=1.0e-10,verbosity=50);
  plot(un,value=1,fill=1,wait=1,cmm="solution by GMRES-RAS",dim=3);
  un = un - uglob;
  plot(un,value=1,fill=1,wait=1,cmm="error of GMRES-RAS",dim=3);
 }
 else {
   un = 0.0;
   LinearCG(opA, un[], rhsglobal[], nbiter=400, precon=opScale,
            eps=1.0e-10,verbosity=50);

   un = 0.0;
   LinearCG(opA, un[], rhsglobal[], nbiter=200, precon=opASM,
            eps=1.0e-10,verbosity=50);

   un = 0.0;
   LinearCG(opA, un[], rhsglobal[], nbiter=200, precon=opASM2,
            eps=1.0e-10,verbosity=50);

   un = 0.0;
```

```
    LinearCG(opA, un[], rhsglobal[], nbiter=200, precon=opASMQ,
             eps=1.0e-10,verbosity=50);

    plot(un,value=1,fill=1,wait=1,cmm="solution by CG-ASM",dim=3);
    un = un - uglob;
    plot(un,value=1,fill=1,wait=1,cmm="error of CG-ASM",dim=3);
}
```

```
// example 7 : NS-cylinder.edp   [slide page 60]
// Navier-Stokes flow around a cylinder
// P2/P1 element with Characteristic Galerkin
// for RIIT Tutorial at Kyushu University, 25 Nov.2016, Atsushi Suzuki

int n1 = 30;
int n2 = 60;
real nu = 1.0/400.0;
real dt = 0.05;
real alpha = 1.0/dt;
int timestepmax = 400;

border ba(t=0,1.0){x=t*10.0-1.0;y=-1.0;label=1;};
border bb(t=0,1.0){x=9.0;y=2.0*t-1.0;label=2;};
border bc(t=0,1.0){x=9.0-10.0*t;y=1.0;label=3;};
border bd(t=0,1.0){x=-1.0;y=1.0-2.0*t;label=4;};
border cc(t=0,2*pi){x=cos(t)*0.25+0.75;y=sin(t)*0.25;label=5;};
mesh Th=buildmesh(ba(n2)+bb(n1)+bc(n2)+bd(n1)+cc(-n1));
plot(Th);
fespace Xh(Th,[P2,P2,P1]);
fespace Vh(Th,P2);
fespace Qh(Th,P1);


Xh [u1,u2,p], [v1,v2,q];
Vh up1, up2;
Qh pp;
macro d11(u1)     dx(u1) //
macro d22(u2)     dy(u2) //
macro d12(u1,u2)  (dy(u1) + dx(u2))/2.0 //
macro div(u1,u2)  (dx(u1) + dy(u2)) //

Qh psi,phi;

func stinflow=y-y*y*y/3.0;
problem streamlines(psi,phi,solver=UMFPACK) =
      int2d(Th)( dx(psi)*dx(phi) + dy(psi)*dy(phi))
   +  int2d(Th)( phi*(dy(u1)-dx(u2)))
   +  on(1,psi=(-2.0/3.0))
   +  on(4,psi=stinflow)
   +  on(3,psi=(2.0/3.0))
   +  on(5,psi=0.0);

streamlines;
plot(psi,wait=1);

problem Stokes([u1,u2,p],[v1,v2,q],solver=UMFPACK) =
  int2d(Th)(
         + 2.0*nu * (d11(u1)*d11(v1)+2.0*d12(u1,u2)*d12(v1,v2)+d22(u2)*d22(v2))
         - p * div(v1, v2) - q * div(u1, u2)) //
+ on(1,3,u2=0)
+ on(4,u1=1.0-y*y,u2=0)
+ on(5,u1=0,u2=0);

int i;
problem NS([u1,u2,p],[v1,v2,q],solver=UMFPACK,init=i) =
  int2d(Th)( alpha * (u1*v1 + u2*v2)
         + 2.0*nu * (d11(u1)*d11(v1)+2.0*d12(u1,u2)*d12(v1,v2)+d22(u2)*d22(v2))
         - p * div(v1, v2) - q * div(u1, u2))
- int2d(Th)( alpha * (convect([up1,up2],-dt,up1)*v1
                     +convect([up1,up2],-dt,up2)*v2) )
+ on(1,3,u2=0)
+ on(4,u1=1.0-y*y,u2=0)
+ on(5,u1=0,u2=0);

plot(Th,wait=1);

Stokes;
for (i = 0; i < timestepmax; i++) {
   up1 = u1;
```

```
    up2 = u2;
    pp = p;
    NS;
    streamlines;
    plot(psi, nbiso=30,wait=0);
    if (i % 20 == 0) {
      plot([up1,up2],pp,wait=0,value=true,coef=0.1);
    }
}
```

```
// example 8 : RayleighBenard.edp   [slide page 69]
// Rayleigh-Benard thermal convection in a box
// P2/P1/P2 element with Characteristic Galerkin
// time evolution data will be stored in "rb.data" for Rayleigh-Benard-stat.edp
// for RIIT Tutorial at Kyushu University, 25 Nov.2016, Atsushi Suzuki

int n1 = 80;
int n2 = 20;
real Pr = 0.71;
real Ra = 1500.0;
real dt = 0.01;
real alpha = 1.0/(dt * Pr);
int timestepmax = 600;

mesh Th=square(n1,n2,[x*4.0,y]);

fespace Xh(Th,[P2,P2,P1]);
fespace Vh(Th,P2);
fespace Qh(Th,P1);

Xh [u1,u2,p], [v1, v2, q];
Vh up1, up2, th, thp, psi;
Qh pp, ss, rr;
macro d11(u1)     dx(u1) //
macro d22(u2)     dy(u2) //
macro d12(u1,u2)  (dy(u1) + dx(u2))/2.0 //
macro div(u1,u2)  (dx(u1) + dy(u2)) //

real epsln = 1.0e-6;     // penalization parameter to avoid pressure ambiguity
int i;
problem NS([u1,u2,p],[v1,v2,q],solver=UMFPACK,init=i) =
  int2d(Th)(alpha * (u1*v1 + u2*v2)
         + 2.0* (d11(u1)*d11(v1)+2.0*d12(u1,u2)*d12(v1,v2)+d22(u2)*d22(v2))
         - p * div(v1, v2) - q * div(u1, u2)
         - p * q * epsln)
- int2d(Th)( alpha * (convect([up1,up2],-dt,up1)*v1
                     +convect([up1,up2],-dt,up2)*v2) )
- int2d(Th) (Ra * thp * v2)
+ on(1,3,u2=0)
+ on(2,4,u1=0);

problem Heat(th,psi,solver=UMFPACK,init=i) =
   int2d(Th)(alpha * (th * psi)
          + dx(th) * dx(psi) + dy(th) * dy(psi))
 - int2d(Th)(alpha * convect([up1, up2], -dt, thp) * psi)
 + on(1,th=1)
 + on(3,th=0);

problem streamlines(ss,rr,solver=UMFPACK) =
      int2d(Th)( dx(ss)*dx(rr) + dy(ss)*dy(rr))
   +  int2d(Th)( rr*(dy(u1)-dx(u2)))
   +  on(1,2,3,4,ss=0.0);

plot(Th,wait=1);

u1[] = 0.0;    // impulsive start
th = (1.0-y); // conductive solution
for (i = 0; i < timestepmax; i++) {
   up1 = u1;
   up2 = u2;
   pp = p;
   thp = th;
   NS;
   Heat;
   plot(th,value=true);
   streamlines;
   plot(ss,nbiso=30);
   if (i % 20 == 0) {
     plot ([u1,u2],value=true,wait=0,coef=0.1);
   }
```

```
}
// write time-evolution data for RayleighBenrad-stat.edp
up1=u1;
up2=u2;
pp=p;
thp=th;
{
    ofstream file("rb.data", binary);
    file.precision(16);
    for (int i = 0; i < up1[].n; i++) {
       file << up1[](i) << " ";
    }
    file << endl;
    for (int i = 0; i < up2[].n; i++) {
      file << up2[](i) << " ";
    }
    file << endl;
    for (int i = 0; i < pp[].n; i++) {
      file << pp[](i) << " " ;
    }
    file << endl;
    for (int i = 0; i < thp[].n; i++) {
       file << thp[](i) << " ";
    }
    file << endl;
}
```

```
// example 9 : RayleighBenard-stat.edp   [slide page 70]
// Navier-Stokes flow around a cylinder
// P2/P1 element with Characteristic Galerkin
// initial data for Newton iteration needs to be prepared in "rb.data"
// by Rayleigh-Benard-stat.edp
// for RIIT Tutorial at Kyushu University, 25 Nov.2016, Atsushi Suzuki

int n1 = 80;
int n2 = 20;
real Pr = 0.71;
real Ra = 1500.0;
int timestepmax = 600;

mesh Th=square(n1,n2,[x*4.0,y]);

fespace Wh(Th,[P2,P2,P1,P2]);
fespace Vh(Th,P2);
fespace Qh(Th,P1);

Wh [u1,u2,p,th], [v1, v2, q, psi];
Vh up1, up2, thp;
Qh pp, ss, rr;
macro d11(u1)     dx(u1) //
macro d22(u2)     dy(u2) //
macro d12(u1,u2)  (dy(u1) + dx(u2))/2.0 //
macro div(u1,u2)  (dx(u1) + dy(u2)) //
macro ugrad(u1,u2,v) (u1*dx(v) + u2*dy(v)) //
macro Ugrad(u1,u2,v1,v2) [ugrad(u1,u2,v1), ugrad(u1,u2,v2)]//

real epsln = 1.0e-6;     // penalization parameter to avoid pressure ambiguity

varf vDRB ([u1,u2,p,th],[v1,v2,q,psi]) =
    int2d(Th)( 2.0*(d11(u1)*d11(v1)+2.0*d12(u1,u2)*d12(v1,v2)+d22(u2)*d22(v2))
              - p * div(v1, v2) - q * div(u1, u2)
              - p * q * epsln
              + (Ugrad(u1,u2,up1,up2)'*[v1,v2] // '
              +  Ugrad(up1,up2,u1,u2)'*[v1,v2]) / (2.0 *  Pr) //'
              )
  - int2d(Th)( Ra * th * v2 )
  + int2d(Th)( dx(th) * dx(psi) + dy(th) * dy(psi) )
  + int2d(Th)( ugrad(up1,up2,th)*psi + ugrad(u1,u2,thp)*psi)
  + on(1,3,u2=0)
  + on(2,4,u1=0)
  + on(1,th=0)
  + on(3,th=0);

// [up1, up2, pp] are obtained from the previous step
varf vRB ([u1,u2,p,th],[v1,v2,q,psi]) =
    int2d(Th)( 2.0*(d11(up1)*d11(v1)+2.0*d12(up1,up2)*d12(v1,v2)+
                    d22(up2)*d22(v2))
              - pp * div(v1, v2) - q * div(up1, up2)
              - pp * q * epsln
              + Ugrad(up1,up2,up1,up2)'*[v1,v2] / (2.0 * Pr) ) //'
  - int2d(Th)( Ra * thp * v2 )
  + int2d(Th)( dx(thp) * dx(psi) + dy(thp) * dy(psi) )
  + int2d(Th)( ugrad(up1,up2,thp)*psi )
  + on(1,3,u2=0)
  + on(2,4,u1=0)
  + on(1,th=0)  // force homogeneous Dirichlet B.C. to be compatible with
  + on(3,th=0); // Jacobian matrix of Newton iteration

problem streamlines(ss,rr,solver=UMFPACK) =
      int2d(Th)( dx(ss)*dx(rr) + dy(ss)*dy(rr))
   +  int2d(Th)( rr*(dy(up1)-dx(up2)))
   +  on(1,2,3,4,ss=0.0);

plot(Th,wait=1);
// read stationary data computed by RayleighBenrad.edp
{
  ifstream file("rb.data", binary);
```

```
  for (int i = 0; i < up1[].n; i++) {
    file >> up1[](i);
  }
  for (int i = 0; i < up2[].n; i++) {
    file >> up2[](i);
  }
  for (int i = 0; i < pp[].n; i++) {
    file >> pp[](i);
  }
  for (int i = 0; i < thp[].n; i++) {
    file >> thp[](i);
  }
}

[u1, u2, p, th] = [up1, up2, pp, thp];
plot(th,cmm="temperature is read from the file",value=true, wait=1);
cout << "start newton" << endl;
for (int n = 0; n < 20; n++) {
  up1 = u1;
  up2 = u2;
  pp = p;
  thp = th;
  plot(thp,cmm="newton n="+n,wait=0,value=true);
  real[int] b = vRB(0, Wh);
  cout << "||b||_l2 " << b.l2 << endl;
  matrix A = vDRB(Wh, Wh, solver=UMFPACK);
  real[int] w = A^-1 * b;
  u1[] -= w;
  cout << "iter = "<< n << "  " << w.l2 << endl;
  if (w.l2 < 1.e-6) break;
}
plot(thp,value=true);
up1 = u1;
up2 = u2;
streamlines;
plot(ss,nbiso=30);
```