FREEFEM++

Marcello Bellomi

Università di Verona

18 Aprile 2013

Indice

Part I

Indroduzione

Dettagli iniziali

- Risolve problemi in 2D e 3D, creato principalmente per risolvere problemi variazionali
- Free software
- Basato su C++, ne mantiene una sintassi dei comandi simile
- Sviluppatori: Frédéric Hecht, Olievier Pironneau, Antoine Le Hyaric, Jacques Morice

Vantaggi

- Generazione/caricamento di mesh
- Possibilità di utilizzare diversi tipi di elementi finiti
- Sono inclusi molti solutori lineari e librerie varie: CG, GMRES, UMFPACK, SUPERLU, ...
- Possibilità di lavorare con matrici sparse
- Sintassi semplice per scrivere la formulazione debole di un particolare problema
- Mailing list efficiente

Svantaggi

- Visualizzazione dei risultati non professionale
- Manuale non sempre chiarissimo
- La documentazione degli esempi sul sito non è organizzata
- Installazione su Linux difficoltosa
- Sintassi particolare per alcune operazioni $+, -, *, /$ vettoriali e scalari

Part II

Esempio base

Equazione di Poisson

Prendiamo un semplice problema differenziale:

$$\begin{cases} -\Delta u(x,y) = f(x,y) & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases}$$

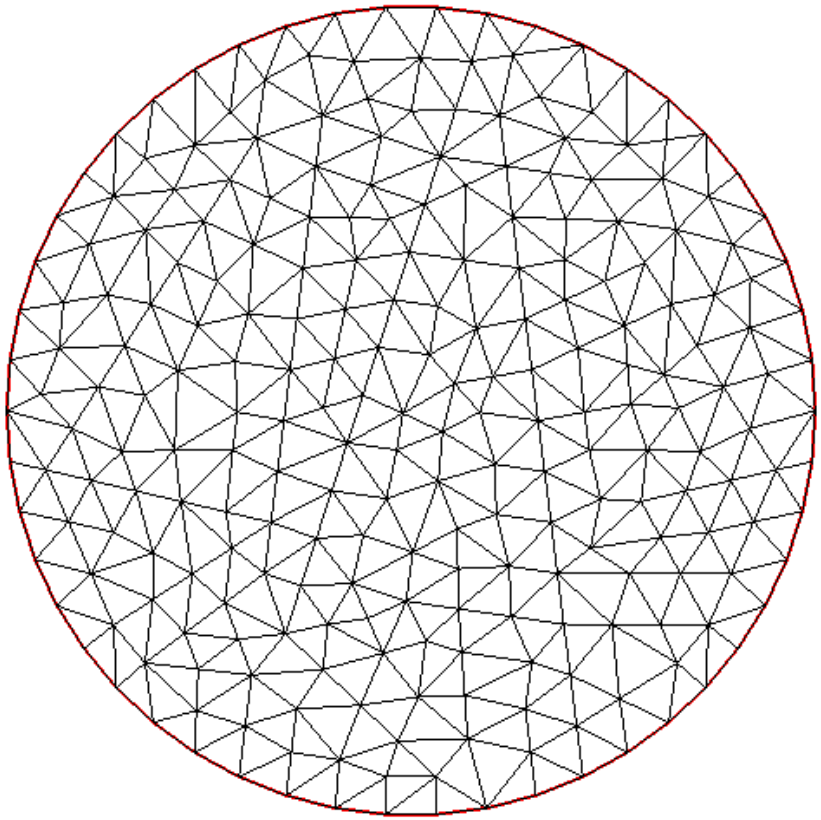
per $f(x,y)$ data, $\Omega \subset \mathbb{R}^2$ aperto e $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$.
Data $f(x,y) = xy$ e sia Ω il disco unitario centrato nell'origine, supponiamo ora di voler risolvere il problema di Poisson sopracitato utilizzando FreeFem++.
Utilizzando un'approccio standard per il calcolo delle variazioni, ricordiamo che la formulazione variazionale in analisi è la seguente:

$$\int_\Omega \nabla u \nabla v \mathrm{d}x\mathrm{d}y = \int_\Omega f v \mathrm{d}x\mathrm{d}y \quad \forall v \in H_0^1(\Omega)$$

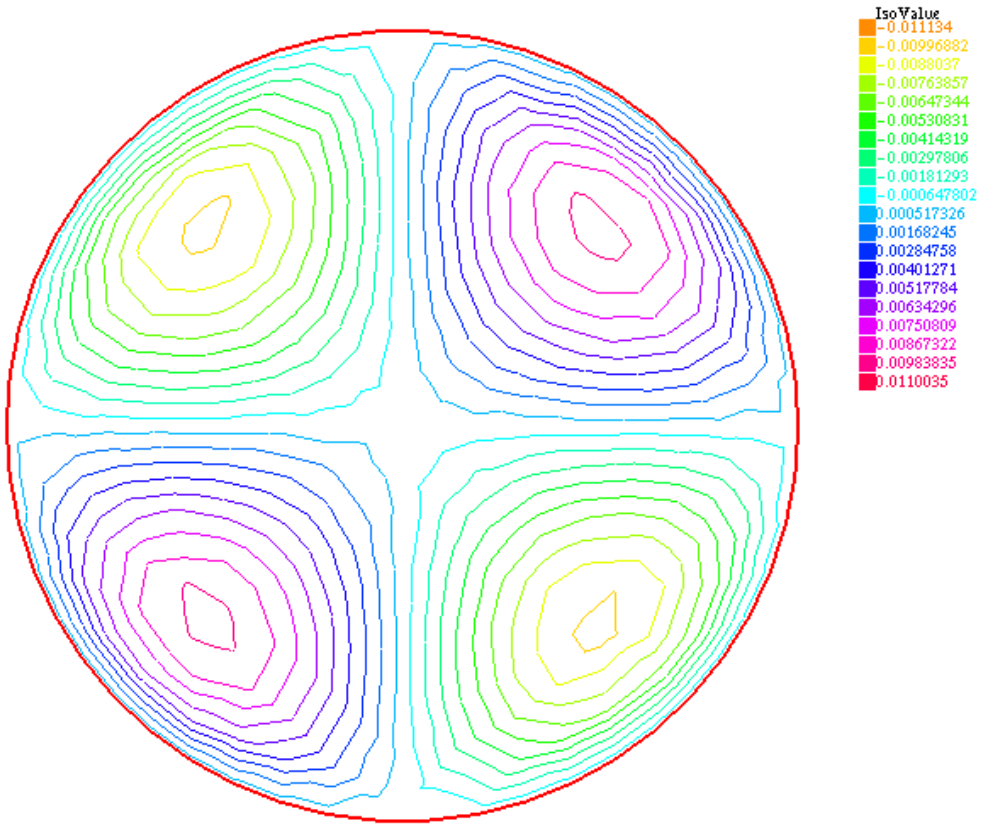
Il primo programma

```
1  // defining the boundary
2  border C(t=0,2*pi){x=cos(t); y=sin(t);}
3  // the triangulated domain Th is on the left side of its boundary
4  mesh Th = buildmesh (C(50));
5  // the finite element space defined over Th is called here Vh
6  fespace Vh(Th,P1);
7  Vh u,v; // defines u and v as piecewise-P1 continuous functions
8  func f= x*y; // definition of a called f function
9  real cpu=clock(); // get the clock in second
10 solve Poisson(u,v,solver=LU) = // defines the PDE
11         int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v)) // bilinear part
12         - int2d(Th)( f*v) // right hand side
13         + on(C,u=0) ; // Dirichlet boundary condition
14 plot(u);
15 cout << " CPU time = " << clock()-cpu << endl;
```

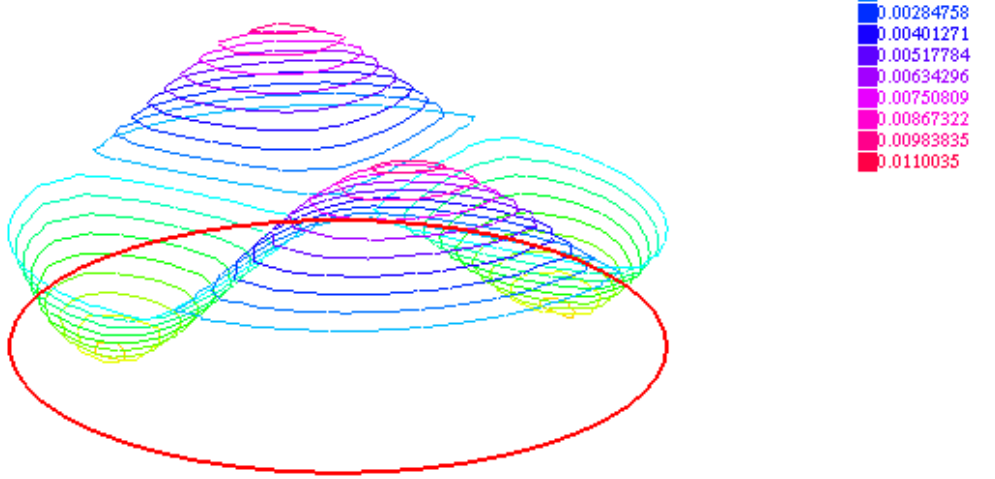
Mesh



Plot


IsoValue
-0.011134
-0.00996882
-0.0088037
-0.00763857
-0.00647344
-0.00530831
-0.00414319
-0.00297806
-0.00181293
-0.000647802
0.000517326
0.00168245
0.00284758
0.00401271
0.00517784
0.00634296
0.00750809
0.00867322
0.00983835
0.0110035

Plot 3D


IsoValue
-0.011134
-0.00996882
-0.0088037
-0.00763857
-0.00647344
-0.00530831
-0.00414319
-0.00297806
-0.00181293
-0.000647802
0.000517326
0.00168245
0.00284758
0.00401271
0.00517784
0.00634296
0.00750809
0.00867322
0.00983835
0.0110035

Part III

Sintassi

Comandi per lavorare con le mesh

Il comando più generale che abbiamo visto è *builtmesh* che agisce su un tipo di dato *border*. Così per creare una mesh si possono scrivere i seguenti semplici comandi:

```
1  border C1(t=t_i,t_f) { x=f(t); y=g(t); label=1; }
2  ...
3  border CN(t=t_i,t_f) { x=f(t); y=g(t); label=N; }
4  mesh Th = builtmesh (C1(nnode_1)+...+CN(nnode_N));
```

Altrimenti, se si cerca di creare una mesh su un quadrato, si può utilizzare un comando più rapido *square*.

Comandi per definire gli spazi degli elementi finiti

Le funzioni che sono coinvolte nel problema devono essere ovviamente trattate come particolari elementi di certi spazi di elementi finiti. Così possiamo definire un tipo di dato *fespace* e specificare che tipo di spazio di elementi finiti si tratta:

$$\text{fespace IDspace(IDmesh,IDFE).}$$

I tipi di elementi finiti che possiamo scegliere sono svariati: **P0**, **P03d**, **P1**, **P13d**, **P1dc**, ...

Comandi per la visualizzazione dei risultati

Il comando basilare per visualizzare mesh e funzioni appartenenti agli elementi finiti è *plot*.
Per visualizzare meglio i risultati ottenuti è possibile settare dei parametri opzionali, i più utili sono i seguenti:

- *dim*
- *value*
- *fill*
- *wait*
- *boundary*

Comandi per creare cicli

I cicli *for* e *while* sono disponibili in FreeFem++ entrambi con le parole chiave *break* e *continue*.
La sintassi è simile a tutti gli altri linguaggi ad alto livello:

$$\text{for}(\text{INIZIALIZZAZIONE; CONDIZIONE; CAMBIAMENTO})\{\text{CORPO}\}$$

Abbiamo poi a disposizione un ciclo per vero con la seguente sintassi:

$$\text{while}(\text{CONDIZIONE})\{\text{CORPO}\}$$

Il comando *solve*

Nell'esempio che abbiamo mostrato abbiamo risolto la formulazione debole utilizzando il comando *solve*.
Il tipo di risolutore utilizzato di default è lo *sparsesolver* che si applica a delle generiche formulazioni deboli che abbiano dipendenza lineare dall'incognita. La soluzione verrà poi calcolata con una tolleranza di 10^{-6}.
C'è la possibilità di fissare inoltre dei parametri opzionali che modificano il modo in cui si risolve il sistema lineare associato alla formulazione variazionale in analisi, tre parametri fondamentali sono:

- *solver*
- *eps*
- *tgv*
- *nbiter*

Comandi di scrittura e lettura da file

Esistono dei particolari tipi di dato riservati per la dichiarazione di file di input o di output.
Per la scrittura su file:

```
1  {
2  ofstream g("nomefile.txt");
3  g << IDvar;
4  };
```

Mentre per la lettura da file:

```
1  {
2  ifstream f("nomefile.txt");
3  f >> IDvar;
4  }
```

Part IV

Esempio

Equazione di Poisson

In questo secondo esempio vedremo due modi per risolvere un equazione di Poisson simile a quella vista nella prima parte, cioè

$$\begin{cases} -\Delta u(x,y) = f(x,y) & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases}$$

Scegliamo $\Omega = [0,1] \times [0,1]$ e $f(x,y) = 2x - 2x^2 + 2y - 2y^2$, in questo modo sappiamo che la soluzione esatta è

$$u(x,y) = x(1-x)y(1-y).$$

Codice per confrontare i due approcci

```
1  // -Delta u = f + homogeneous Dirichlet b.c.
2  func f = 2*x-2*x^2+2*y-2*y^2;
3  int nnode = 17;
4  mesh Th = square(nnode,nnode);
5  fespace Vh(Th,P2);
6  Vh u,v,F,G;
7  u = x*(1-x)*y*(1-y); // exact solution
8  //First way
9  Vh vv,phi;
10 solve Poisson(vv,phi,solver=GMRES) =
11         int2d(Th)(dx(vv)*dx(phi) + dy(vv)*dy(phi)) // bilinear part
12         - int2d(Th)(f*phi) // right hand side
13         + on(1,2,3,4,vv=0) ; // Dirichlet boundary condition
14 cout << "*** error SOLVE " << sqrt(int2d(Th)(abs(u-vv)^2)) << endl;
15 //Second way
16 int n = Vh.ndof;
17 varf rhs (unused,v) = int2d(Th)(f*v)+on(1,2,3,4,unused=0);
18 F[] = rhs(0,Vh);
19 real[int] res(n);
20 func real[int] Atimesxx(real[int] & xx)
21 {
22     Vh xxloc;
23     xxloc[] = xx;
24     varf bilinear (unused,v) = int2d(Th)(dx(xxloc)*dx(v)+dy(xxloc)*dy(v))+on(1,2,3,4,unused=xxloc);
25     res = bilinear(0,Vh,tgv=1);
26     return res;
27 }
28 v = 0;
29 LinearGMRES(Atimesxx,v[],F[],nbiter=200);
30 cout << "*** error GMRES " << sqrt(int2d(Th)(abs(u-v)^2)) << endl;
```

Il comando *varf*

Il comando *varf* identifica uno dei principali tipi di dato e serve per definire una particolare forma variazionale piena.
La sintassi standard è la seguente:

```
1   varf IDvarform (u,v) = int2d(Th)( argomento dipendende da u e v );
```

Come abbiamo visto nell'esempio precedente è possibile definire una forma variazionale dipendente solo da un input, ciò si realizza con il comando *unused*. E' possibile inoltre fissare delle condizioni di bordo solo sul primo input e ciò si effettua utilizzando la stessa sintassi che abbiamo visto con il comando *solve*.

Il comando *LinearGMRES*

Il comando in analisi serve per risolvere problemi variazionali e per risolvere sistemi lineari utilizzando l'algoritmo GMRES (Gereralized Minimum Residual).
La sintassi basilare per risolvere sistemi lineari è questa:

```
1   LinearGMRES(A,x,b);
```

dove vogliamo risolvere il sistema $Ax = b$. Bisogna fare attenzione però perchè l'utilizzo del vettore x ha un duplice ruolo: come input rappresenta il vettore di partenza per l'algoritmo GMRES e come output perchè il valore di partenza viene sovrascritto e su x viene salvata la soluzione del sistema lineare.

Il comando *LinearGMRES*

Vediamo ora come utilizzare il *LinearGMRES* per risolvere una particolare formulazione variazionale. Supponiamo di dover trovare u tale che

$$a(u, v) = \langle f, v \rangle \quad \forall v \in Vh$$

dove $a(\cdot, \cdot)$ è una forma bilineare e coerciva.
Per trovare tale u si può scrivere:

```
1  LinearGMRES(Atimesxx,v[],F[]);
```

dove *Atimesxx* è una particolare func che dato in input un vettore *xx* restituisce il prodotto matrice-vettore tra la matrice associata alla forma bilineare $a(\cdot, \cdot)$ e il vettore *xx*, dove F è la forma variazionale associata al termine noto $\langle f, v \rangle$.

FINE