Continuum Modeling: Numerical schemes for linear & nonlinear reaction/diffusion systems, Phase fields Modeling
Part 3: Numerical solution with Freefrem++ (a simple introduction)

Jean-Paul CHEHAB

LAMFA UMR CNRS 7352, Univ. Picardie Jules Verne

September 11-12th 2017
SiMBioS, Universidade da Madeira, September 11-15, 2017 Stochastic and Deterministic Mathematical Methods for Biological and Environmental Systems: Theory Applications

Plan

1. Goal

Plan

Plan

1. Goal
2. Basic structure of the code
3. Starting with FreeFEM++

Plan

1. Goal
2. Basic structure of the code
3. Starting with FreeFEM++
4. Solution of elliptic problem (Poisson)
   - Build a mesh

Plan

Plan

Plan

FreeFEM++

is a Free software to solve PDE using the Finite Element Method.
It runs on most of available systems : Windows, Linux and MacOS

What FreeFEM++ does

- automatic Mesh Generation
- automatic building of mass and stiffness matrices (taking into account BC)
- Solution of discrete linear systems
- Allows to simulate 2D and 3D problems in many fields such as CFD
- Post-Treatment facilities (Graphics, Text, File generation)

Here a nice basic tutorial
http://homepage.ntu.edu.tw/ twhsheu/twsiamff++/tutorials/2014-Casts/ff-basic-tutorial.pdf

- Definition of the geometry then of the mesh
- Definition of the FEM space
- Solution of the variational problem
- Post-treatment (graphics ...)

Download the software at `http://freefem/org/f++`

Use

- Write the script using a raw editor
- Save it with the file extension `.edp`
- Run it

What is needed

- Know what a variational formulation is
- (better) Know a little from C++

Rectangle

```
int m=10, n=10;
mesh Th=square(m,n, [x,y]);
plot(Th,wait=1,cmm="The rectangle") ;
```

Disk

```
int m=30;
border C1(t=0 ,2* pi){x=2*cos (t);y=2*sin (t); label =1;};
mesh Th=buildmesh (C1(m));
plot(Th,wait=1,cmm="A disk") ;
```

Torus

```
int m1=200,m2=100;
border C1(t=0 ,2* pi){x=2*cos (t);y=2*sin (t); label =1;};
border C2(t=0 ,2* pi){x=cos (t);y=sin (t); label =1;};
mesh Th=buildmesh(C1 (m1)+C2(-m2));
plot(Th,wait=1,cmm="The Torus") ;
```

Generate Finite Element Space

- The most simple
  `fespace Vh(Th,P1);`
- But also other P-elements
  `fespace Vh(Th,P2);`
  and
  `fespace Vh(Th,P3);`

We look to $u_h \in V_h \subset V$:

$$(\mathcal{V}_h) : \int_\Omega \nabla u_h \nabla \mathbf{v}_h dx - \int_\Omega f \mathbf{v}_h dx = 0 \forall \mathbf{v}_h \in V_h$$

Here $V_h = \{\mathbf{v}_h : (\mathbf{v}_h)|_T \in \Pi_1, \text{ for any } T \in T_h\}$ where

- T_h is a regular mesh of Ω
- Π_1 is the space of polynomials whose degree is lower or equal to one

FreeFem Structure of the problem

```
problem Poisson(u,v)=// Definition of the problem
int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))// bilinear form
-int2d(Th)(f*v)// linear form
+on(1,2,3,4,u=0); // Dirichlet Conditions
Poisson; // Solve Poisson Equation
```

Exercise 1

A first example, build a FreeFem++ code for solving Poisson problem with Homogeneous Dirichlet Boundary conditions

Solution

```
mesh Th= square(10,10); // mesh generation of a square
fespace Vh(Th,P1); // space of P1 Finite Elements
Vh u,v; // u and v belong to V_h
func f=cos(x)*y; // f is a function of x and y
problem Poisson(u,v)=// Definition of the problem
int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))// bilinear form
-int2d(Th)(f*v)// linear form
+on(1,2,3,4,u=0); // Dirichlet Conditions
Poisson; // Solve Poisson Equation
plot(u);// Plot the result
```

Save : save.edp

```
include "Poisson.edp"; // include previous script
plot(u,ps="result.eps") f=cos(x)*y; // Generate .eps output
save mesh(Th,"Th.msh");// Save the Mesh
ofstream file("potential.txt")
file<<u[];
```

Read : Read.edp

```
mesh Th=readmesh("Th.msh"); // Read the Mesh
fespace Vh(Th,P1) ;
Vh u=0;
ofstream file("potential.txt"); // Read the potential
plot(u,cmm"The result was correctly saved:)");
```

Exercise 2

A first example, build a FreFem++ code for solving Poisson problem with Mixed Dirichlet-Neumann Boundary conditions

Solution

```
int Dirichlet=1,Neumann=2; // For label definition
border a(t=0,2.*pi){x=cos(t);y=sin(t);label=Dirichlet;};
border
b(t=0,2.*pi){x=0.2*cos(t)+0.3;y=sin(t)*0.2+0.3;label=Neumann;};
mesh Th=buildmesh(a(80)+b(-20));
fespace Vh(Sh,P1);
Vh u,v;
func f=cos(x)*y; func ud=x; func g=1.;
problem Poisson(u,v)= int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))
+int1d(Th,Neumann)(g*v)
-int2d(Th)(f*v)
+on(Dirichlet,u=ud); // u=ud on label=Dirichlet=1
Poisson; plot(u);
```

Impossible in Batman school not to consider **Robin** Boundary conditions !

Impossible in Batman school not to consider **Robin** Boundary conditions ! They write as

$$au + \kappa \frac{\partial u}{\partial n} = b$$

and they can be implemented as

```
border gammar=...
```

then in the variational problem

```
int1d(Th,gammar)(a*u*v)-int1d(Th,gammar)(b*v).
```

Practically, the Numerical solution resumes as a sequence of solutions of approximated variational problems. It suffices then to use a loop structure : first define the problem to be solve at each step.

Exercise 3

Write a FreeFem++ code `BackwardEuler` for solving the variational problem attached to each step of the Backward Euler Scheme

Solution

```
problem BackwardEuler(u,v)= int2d(Th)(u*v)
int2d(Th)(dt*(dx(u)*dx(v)+dy(u)*dy(v)))
-int2d(Th)(u0*v)
-int2d(Th)(f*v);
```

To go further and iterate in time, we'll use the following looping structure

Loop structure

```
real delta=0.05,T=1; // time step and max time
for(t=0;t<T;t+=dt){
BackwardEuler;
```

Consider the reaction-diffusion equation

$$\frac{\partial u}{\partial t} - \Delta u + f(u) = 0$$

A first semi implicit scheme

At first, we apply a semi-implicit scheme :

$$\frac{u^{(k+1)} - u^{(k)}}{\Delta t} - \Delta u^{(k+1)} + f(u^{(k)}) = 0,$$

to which we associate the variational problem

$$< u^{(k+1)}, v > +\Delta t < \nabla u^{(k+1)}, \nabla v >=< u^{(k)}, v > +\Delta t < f(u^{(k)}), v >$$

The loop structure can be written as

```
for(t=0;t<T;t+=dt){
f=F(u0); BackwardEuler;
u0=u;
}
```

A fully implicit scheme

At first, we apply a semi-implicit scheme :

$$\frac{u^{(k+1)} - u^{(k)}}{\Delta t} - \Delta u^{(k+1)} + f(u^{(k+1)}) = 0,$$

This problem is nonlinear and can be solved numerically by using, e.g., a Picard fixed point method, setting also $a(u, v) =< u, v > +\Delta t < \nabla u, \nabla v >$

Algorithm	Fixed point for fully backward Euler's
Set	$u^{(k,0)} = u^{(k)}$
Compute	residual=30
While	$m < Mmax$ & residual $> \eta$,
Solve	$a(u^{(k,m+1)}, v) =< u^{(k)}, v > +\Delta t < f(u^{(k,m)}), v > \forall v \in V$
Compute residual	residual=$\|u^{(k,m+1)} - u^{(k)} - \Delta t \Delta u^{(k,m+1)} + \Delta t f(u^{(k+1,m)})\|$
EndFor	
Set	$u^{(k+1)} = u^{(k,m+1)}$

A fully implicit scheme : the whole loop

The whole loop structure can be written in FreeFem++ as

```
f=f(u0);
for(k=0;k<Kmax;k+=1){
residual=10;p=0;
while(p<40  residu >0.00000001)){
BackwardEuler;
f=F(u);
p=p+1;
```

$\}u0 = u;$

$\}$

Exercise 4

Write a FreeFem++ code for solving Allen-Cahn equation

The solution of Cahn-Hilliard equation is easier when decoupled in two equations, so the basis of any (semi-)implicit scheme is the solution of a coupled linear system. A nice way to do it, is to generate the block matrix using the command `varf`. We consider for simplicity first only time integration of the linear part of Cahn-Hilliard equation by a Backward Euler's method.

- First define the product of FE spaces
```
fespace Vh2(Th,[P1,P1]);
```
- Define the block matrix attached to the coupled variational problem as
```
varf AA([v,w],[phi,psi]) = int2d(Th) (v*phi/dt)
int2d(Th)(dx(w)*dx(phi)+dy(w)*dy(phi))
+int2d(Th)(dx(v)*dx(psi)+dy(v)*dy(psi)-w*psi );
```
- Building the block matrix
```
H=AA(Vh2,Vh2, factorize=1, solver=LU);
```
- initial datum
```
[v,w]=[u0,0];
```
- Solution of the linear system (source is the right hand side

v[]=H$\wedge - 1 * source$[];