Laboratory Sessions : FreeFem++

Preschool CIMPA Workshop

June 24^{th} – July 4^{th} 2015

IIT Mumbai

IIT Mumbai 400 076

Abstract

The manual will guide you through the session and with some explict step by step instructions to follow during the FreeFem++ Laboratory Sessions.

The Main objective: there is a definitive objective for each session and the Laboratory Sessions will synchronise with the theory being covered in class, e.g, we start with steady state linear problems (elliptic), move to transient simulations (parabolic heat equation), implement the mixed method (with the linear Stokes problem) and end with sessions on the Navier-Stokes equation. Steady State Nonlinear problems and the Adaptive meshing algorithms will also be covered.

The manual is a rich source of useful information, the key is that there are a *huge* number of examples. We learn by *spending time* in looking carefully at these illuminating examples. The manual and some useful documents are available in *documents* folder.

Contents

1 Session 1: Steady State Problem: FEM for Elliptic PDE, e.g., Laplace and Poisson

In this session, we pick up the simplest problem in FEM; an elliptic PDE in two dimensions and also learn how the standard algorithmic steps in a FEM solution are executed with FreeFem++.

§ The complete program to solve an planar Elliptic PDE.

§ Simple exercises to map the program to the FEM algorithm.

§ A few more exercises.

We will solve,

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

The statement of the weak form of the problem after discretization is,
Find $u_h(x,y)$ such that,

$$\int_{T_h} \left(\frac{\partial u_h}{\partial x}\frac{\partial v_h}{\partial x} + \frac{\partial u_h}{\partial y}\frac{\partial v_h}{\partial y} \right) = \int_{T_h} f v_h \tag{1.1}$$

where u_h and v_h belong to suitable finite dimensional subspaces of suitable Hilbert spaces.

Let us see the program in FreeFem++ for this Poisson problem on a circular domain with $f(x,y) = x \times y$

1.1 Getting Started

1.1.1 Execute the complete program in FreeFem++

The program is given below. Copy it

1. Onto your command window.
2. Save it in the Directory *"Contents/Resources/examples/chapt3" as "myfirstprogram.edp"*.
3. Execute it.

```
// defining the boundary
border C(t=0,2*pi){x=cos(t); y=sin(t);}
// the triangulated domain Th is on the left side of its boundar
mesh Th = buildmesh (C(50));
// the finite element space defined over Th is called here Vh
fespace Vh(Th,P1);
Vh u,v; // defines u and v as piecewise-P1 continuous functions
func f= x*y; // definition of a called f function
real cpu=clock(); // get the clock in second
solve Poisson(u,v,solver=LU) = // defines the PDE
int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v)) // bilinear part
- int2d(Th)( f*v) // right hand side
+ on(C,u=0) ; // Dirichlet boundary condition
plot(u);
```

1.1.2 What is happening behind these innocent looking lines of code?

1. We can visualise some of these lines. Study the following lines of code.

```
bool debug = true;
border a(t=0,2*pi){x=cos(t); y=sin(t); label=1;}
border b(t=0,2*pi){x=0.3+0.3*cos(t); y=0.3*sin(t); label=1;}
plot(a(50)+b(-30),wait=debug);//plot the borders to see the intersection
//(so change the 0.8 to say 0.3 and then a mouse click
mesh Th= buildmesh(a(50)+b(-30));
```

```
plot(Th, wait=debug); // Plot Th then needs a mouse click
fespace Vh(Th,P2);
Vh f = sin(pi*x)*cos(pi*y);
plot(f,wait=debug);
Vh g=sin(pi*x+cos(pi*y));
plot(g,wait=debug);
```

2. Now *insert suitably* the lines of code required to *visually* understand the first few lines of the code of "myfirstprogram.edp". You should re-open "myfirstprogram.edp"

3. The next few lines is nothing but the weak form (given in eq(1.1) above) of this particular problem with the Dirichlet boundary condition. Our background in FEM clearly tells us what happens when the program executes this line i.e, it obtains suitably the linear form finite dimensional $Au = b$ and solves (we have chosen the LU solver amongst the many available) it for the piecewise linear u (P_1 finite elements).

4. The final line is just a visual plot of the contours of the isovalues of $u(x, y)$.

5. Open the program *"Contents/Resources/examples/ffcs/p2hat.edp"*. Now add suitable lines to visualize the function $f = x \times y$ in "myprogram.edp". Do the same if the problem is solved with P_2 finite elements. Visualize the solution $u(x, y)$ in the same way (i.e., in 3-d).

1.1.3 Some quick really simple variations

1. Change the domain to an ellipse.

2. Half the mesh size.

3. Try a piecewise quadratic approximation.

1.2 Moving ahead

- Load the file "membrane.edp" from "examples/chap3". Execute it. *What is the boundary condition in this problem*
- Load the file "membranerror.edp" from "examples/chap3". Execute it. *What is the boundary condition in this problem?*.

1.3 Some Illustrative Domains and meshings

Visualise the following illustrative code snippets related to domains and meshing.

```
real x0=1.2,x1=1.8;
real y0=0,y1=1;
int n=5,m=20;
//meshing with different flags. check pg-104 of manual for details
for (int i=0;i<5;++i)
{
int[int] labs=[11,12,13,14];
mesh Th=square(3,3,flags=i,label=labs,region=10);
plot(Th,wait=1,cmm=" square flags = "+i );
}
```

1.4 Some Illustrative Visualizations

1.4.1 Holes

```
border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
border b(t=0,2*pi){ x=0.3+0.3*cos(t); y=0.3*sin(t);label=2;}
plot(a(50)+b(+30)) ;//to see a plot of the border mesh
mesh Thwithouthole= buildmesh(a(50)+b(+30));
mesh Thwithhole= buildmesh(a(50)+b(-30));
plot(Thwithouthole,wait=1,ps="Thwithouthole.eps");
plot(Thwithhole,wait=1,ps="Thwithhole.eps");
/*ps="fileName" is used to generate a postscript file with identification shown on the figure.*/
```

1.4.2 rectangular boundary

```
real x0=1.2,x1=1.8;
real y0=0,y1=1;
int n=5,m=20;
//defining a rectangle with dimensions 5*20 with specified coordinates
mesh Th=square(n,m,[x0+(x1-x0)*x,y0+(y1-y0)*y]);
plot(Th,cmm="a");
//cmm will add the comment to the fig
```

1.4.3 Multiple boundaries

```
int upper = 1;
int others = 2;
int inner = 3;
border C01(t=0,1){x=0;y=-1+t;label=upper;}
border C02(t=0,1){x=1.5-1.5*t;y=-1;label=upper;}
border C03(t=0,1){x=1.5;y=-t;label=upper;}
border C04(t=0,1){x=1+0.5*t;y=0;label=others;}
border C05(t=0,1){x=0.5+0.5*t;y=0;label=others;}
border C06(t=0,1){x=0.5*t;y=0;label=others;}
border C11(t=0,1){x=0.5;y=-0.5*t;label=inner;}
border C12(t=0,1){x=0.5+0.5*t;y=-0.5;label=inner;}
border C13(t=0,1){x = 1;y = -0.5+0.5*t;label = inner;}
int n = 10;
plot(C01(-n)+C02(-n)+C03(-n)+C04(-n)+C05(-n)+C06(-n)+
C11(n)+C12(n)+C13(n), wait=true);
mesh Th = buildmesh(C01(-n)+C02(-n)+C03(-n)+C04(-n)+C05(-n)+C06(-n)+
C11(n)+C12(n)+C13(n));
plot(Th, wait=true);
cout << "Part 1 has region number " << Th(0.75, -0.25).region << endl;
cout << "Part 2 has redion number " << Th(0.25, -0.25).region << endl;
```

1.5 Homework

Essentially there is a lot of useful information in the manual and also in the huge set of *illuminating illustrative* examples. So, please check these key features which will play a ubiquitous role

1.5.1 Finite elements available

Check sections *6.1-6.6*in the manual which is list of finite elements available.

1.5.2 Extracting the matrices and other relevant data important for you

FreeFem++ has two important applications of the keyword *varf*. See section *6.12* for detailed explanations and search this word throughout the manual for several examples.

As a first example you can check by executing the file "test1.edp" from "examples/chap3".

1.5.3 Visualization

See Chapter *7* of the manual.

As a first example you must check a nice demo file "options.edp" from "examples/ffcs".

1.5.4 Parameters affecting solve and problem

See section *6.9* for all the details.

1.5.5 The func operator

The "func" operator is fairly intuitive. See also "func.edp" from "examples/tutorial"

1.5.6 The *convect* Operator

The "convect" will play a central role in the algorithms for *transient* simulation of Stokes and Navier-Stokes. Look it up in both the manual and the examples. *Now you now how to navigate through the wonderful manual and stock of great examples provided by the architects of FreeFem++ !!!! :) :)*

2 Session 2: Evolution Problems: FEM for a Parabolic PDE, e.g., Transient Heat Conduction

We focus on transient heat conduction. The spatial discretization of the Laplacian is by FEM and time is discretized by finite differences. We realise the ease of obtaining the evolution of the solution in FreeFem++. We also use FreeFem++ to carry out the standard text book simulations of the well known explicit and implicit finite difference schemes to map our text book knowledge of stability of these schemes.

§ A transient heat conduction problem.

§ Simulation of two well known finite difference schemes.

§ A few exercises and homework.

In the last session, we got an exposure to FEM for space discretization. Here we focus on the classical finite difference schemes to deal with time marching and combine these to solve the Parabolic heat equation. The details are given by Allaire and a brief is also given in the manual in examples of Chapter 9.

The problem we solve is

$$\begin{aligned} \frac{\partial u}{\partial t} - \mu\Delta u &= f \text{ in } \Omega\times]0,T[, \\ u(\mathbf{x},0) &= u_0(\mathbf{x}) \text{ in } \Omega; \\ \left(\frac{\partial u}{\partial n}\right)(\mathbf{x},t) &= \mathbf{0} \text{ on } \partial\Omega\times]0,T[\end{aligned}$$

Using *backward Euler* the time discretised equation with the notation $u^m(x,y) = u(x,y,m\tau)$ is as follows

$$\begin{aligned} \frac{u^{m+1}-u^m}{\tau} - \mu\Delta u^{m+1} &= f^{m+1} \text{ in } \Omega \\ u^0(\mathbf{x}) &= u_0(\mathbf{x}) \text{ x } \Omega; \\ \frac{\partial u^{m+1}}{\partial n(\mathbf{x})} &= 0 \text{ on } \partial\Omega, \forall m = 0,\ldots,[T/\tau] \end{aligned}$$

and the and the weak form is

$$\int_\Omega \{u^{m+1}v + \tau \nabla u^{m+1} \cdot \nabla v\} - \int_\Omega \{u^m v + \tau f^{m+1} v\} = 0 \tag{2.1}$$

2.1 Heat equation on a square domain

See the following code that calculates u_h^m of u^m in a step by step mannerwith respect to t.

Note: *The following problem has an exact solution* $u(x, y, t) = tx^4$ *and therefore we can calculate* L^2 *error at end of any time step. This is done at the end of the code*

The problem that is solved is

$$\begin{aligned} \frac{\partial u}{\partial t} - \mu \Delta u &= x^4 - \mu 12 t x^2 \text{ in } \Omega \times]0,3[, \quad \Omega =]0,1[\times]0,1[\\ u(\mathbf{x}, 0) &= 0 \text{ on } \Omega; \\ u|_{\partial\Omega} &= t \times x^4 \text{ on } \partial\Omega \times]0,T[\end{aligned}$$

```
mesh Th=square(16,16);
fespace Vh(Th,P1);
Vh u,v,uu,f,g;
real dt = 0.1, mu = 0.01;//Time step and viscosity
problem dHeat(u,v) =
int2d(Th)( u*v + dt*mu*(dx(u)*dx(v) + dy(u)*dy(v)))
+ int2d(Th) (- uu*v - dt*f*v )
+ on(1,2,3,4,u=g); //Boundary condition is ???
real t = 0; // start from t=0
uu = 0; // u(x,y,0)=0, Initial condition
//Loop over time, time marching till T=3
for (int m=0;m<=3/dt;m++)
{
t=t+dt;
f = x^4-mu*t*12*x^2;//Note that f depends on t
g = t*x^4;//g also depends on t
dHeat; // Solve the weak form defined in Problem
```

```
plot(u,wait=true);// Plot the solution at every time step
uu = u;
cout <<"t="<<t<<"L^2-Error="<<sqrt( int2d(Th)((u-t*x^4)^2) ) << endl;
// L2 error is plotted at every time step because analytical solution is known
}
```

In the last statement, the L^2-error $\left(\int_\omega |u - tx^4|^2\right)^{\frac{1}{2}}$ is calculated at $t = m\tau, \tau = 0.1$. Note the error increases with m. The *for loop* is used for the iteration.

2.2 Numerical Experiments on Mathematical theory of Time Difference Approximations

Recall the standard results for time stepping schemes for the following discretized weak form

$$\frac{1}{\tau}\left(u_h^{n+1} - u_h^n, \phi_i\right) + a\left(u_h^{n+\theta}, \phi_i\right) = \left\langle f^{n+\theta}, \phi_i \right\rangle$$
$$i = 1, \ldots, m, \quad n = 0, \ldots, N_T$$
$$\text{where } \; u_h^{n+\theta} = \theta u_h^{n+1} + (1-\theta)u_h^n, \quad f^{n+\theta} = \theta f^{n+1} + (1-\theta)f^n$$

and $\theta \in [0, 1]$.

The well known *stability* result is that the schemes are unconditionally stable when $1/2 \leq \theta \leq 1$ and conditionally stable otherwise.

Use the code below to see the stability results at different θ. Please note that the weak form given above is implemented in this code. You should be easily able to make out the boundary conditions being used in this problem.

Note: the exact solution for this problem is $tx^2 \times (1-x)^2$. The error is FEsolution-exact. The L_∞ norm is being calculated. This error is being normalised by the L_∞ norm of the exact solution i.e., $||err||_{L^\infty(\Omega)}/||exact||_{L^\infty(\Omega)}$

```
mesh Th=square(12,12);
fespace Vh(Th,P1);
fespace Ph(Th,P0);
Ph h = hTriangle; // mesh sizes for each triangle
real tau = 0.1, theta=0.; //Time step and initial value of theta
func real f(real t) {
return x^2*(x-1)^2 + t*(-2 + 12*x - 11*x^2 - 2*x^3 + x^4);
```

```
}
//the function f(t,x)
ofstream out("err02.csv"); // file to store calculations
out << "mesh size = "<<h[].max<<", time step = "<<tau<<endl;
for (int n=0;n<5/tau;n++)
out<<n*tau<<",";
out << endl;
Vh u,v,oldU;
Vh f1, f0;
problem aTau(u,v) =
int2d(Th)( u*v + theta*tau*(dx(u)*dx(v) + dy(u)*dy(v) + u*v))
- int2d(Th)(oldU*v - (1-theta)*tau*(dx(oldU)*dx(v)+dy(oldU)*dy(v)+oldU*v))
- int2d(Th)(tau*( theta*f1+(1-theta)*f0 )*v );//Weak form of all time schemes
//We loop on theta in steps of 0.1 to spot the unstability point
while (theta <= 1.0) {
real t = 0, T=3; // from t=0 to T
oldU = 0; // u(x,y,0)=0, Initial condition
out <<theta<<",";
for (int n=0;n<T/tau;n++) {
t = t+tau;
f0 = f(n*tau); f1 = f((n+1)*tau);
aTau;
oldU = u;
plot(u);
Vh uex = t*x^2*(1-x)^2; // exact sol.= tx2(1  x)2
Vh err = u - uex; // err =FE-sol - exact
out<< abs(err[].max)/abs(uex[].max) <<",";
//L2 norm (normalised by exact solution
}
out << endl;
theta = theta + 0.1;
}
```

You will see that θ becomes unstable at $\theta = 0.4$. Please see the plot Figure *9.19* in the FreeFem++ manual.

2.3 Two nice illustrations from Florian De Vuyst

See section 3.2 (Heat.edp) and a real problem from Engineering in Section 3.3 (ThermalDesign.edp)

2.4 Homework

Look up the *convect* and *varf* and *func* operator in FreeFem++. The *convect* operator is rucial to the examples in Florian De Vuyst. So, there is a good explanation in Chapter 4 of this manual.

3 Session 3: Nonlinear Steady State Problems: FEM and Newton Raphson, e.g., Steady State Navier-Stokes

In this session, the aim is to demonstrate the steps by which FreeFem++ obtains the steady state solution of a Nonlinear PDE. The key steps are the linearization of the weak form of the equation and the Newton-Raphson iteration. We illustrate these key steps in FreeFem++ via the example of Steady State solution of the Navier-Stokes equation.

§ The Newton-Raphson iteration.

§ The Navier-Stokes: a weak form and linearization.

§ An important tip: How do I get a *good guess* for a complicated problem in Nonlinear Steady State problems.

3.1 The Newton-Raphson iteration

We summarise this ubiquitous algorithm used to solve nonlinear problems with iteration. Find $u \in V$ such that $F(u) = 0$ where $F : V \mapsto V$

- Choose $u_0 \in \mathbb{R}^n$. (An initial hiccup, we need an starting guess)
- for (i=0, number of iterations, i=1+1)
  1. Solve for δu at a *known value of* u_i, the linear equation $DF(u_i)\delta u = F(u_i)$
  2. $u_{i+1} = u_i - w_i$

  break when $||w_i|| < \epsilon$

where $DF(u)$ is the differential of F at the point u, this is a linear application such that $F(u + \delta) = F(u) + DF(u)\delta + o(\delta)$

3.2 Quick review of Linearization formulae

Refer to Chapter 2 on differentiation in the book *Analysis on Manifolds* by James Munkres.

Definition: Let $A \subset \mathbb{R}^n$, let $f : A \longrightarrow \mathbb{R}^n$. Suppose A contains a neighbourhood of u. We say f is *differentiable at a* if there is a $ntimesm$ matrix B such that

$$\frac{f(\mathbf{u}+\mathbf{h}) - f(\mathbf{u}) - B\mathbf{h}}{|h|} \rightarrow \mathbf{0} \ \text{as}\mathbf{h} \rightarrow \mathbf{0}$$

This matrix B is unique and called the derivative of f at u and is denoted by $Df(u)$.

Given $\delta u \in \mathbb{R}^m$ with $u \neq 0$, define

$$f'(\mathbf{u}; \delta\mathbf{u}) = \lim_{t\to 0} \frac{f(\mathbf{a}+t\mathbf{u} - f(\mathbf{a})}{t}$$

Theorem: If f is differentiable at $\mathbf{u}$, then all the directional derivatives of f at $\mathbf{u}$ exist, and

$$Df(\mathbf{u})\delta\mathbf{u} = f'(\mathbf{a}; \mathbf{u})$$

For analogous results for Banach spaces, see Chapter 4 of the book by Zeidler, Functional Analysis, Main Principles and their applications.

In particular, the following result on results analogous to *partial derivatives* has a simple short proof in the book,

Theorem Let the map $f : U(u, v) \subseteq X \times Y \rightarrow Z$ be given on an open neighbourhood of the point (u, v), where X, Y and Z are Banach spaces over $\mathbf{R}$, if the derivative $f'(u, v)$ exists, then the partial derivatives $f_u(u, v)$, $f_v(u, v)$ also exist and

$$f'(u, v)(h, k) = f_u(u, v)h + f_v(u, v)k \ \ \forall\ h \in X\ , k \in Y$$

3.3 Example of Steady Navier-Stokes equation: NSNewton.edp

3.3.1 Strong form

Find the velocity field u_1, u_2 and the pressure p of a flow in the domain $\Omega \subset \mathbb{R}^2$ such that

$$\begin{aligned} (\mathbf{u}\cdot\nabla)\mathbf{u} - \nu\Delta\mathbf{u} + \nabla p &= \mathbf{0} \\ \nabla \cdot u &= \mathbf{0} \end{aligned}$$

3.3.2 Weak Form

Find $\mathbf{u}, p$ such that $\forall \mathbf{v}$ (zero on Γ) and $\forall q$

$$\int_\Omega ((\mathbf{u}\cdot\nabla)\mathbf{u}).v + \nu\nabla\mathbf{u} : \nabla\mathbf{v} - p\nabla\cdot\mathbf{v} - q\nabla\cdot\mathbf{u} - 1e^{-}8 \times q \times p = 0$$

3.3.3 Weak Form and Linearizarion

Thus,

$$\begin{aligned} F(\mathbf{u},p) &= \int_\Omega ((\mathbf{u}\cdot\nabla)\mathbf{u}).v + \nu\nabla\mathbf{u} : \nabla\mathbf{v} - p\nabla\cdot\mathbf{v} - q\nabla\cdot\mathbf{u} \\ &\quad -1e^{-}8 \times q \times p \\ DF(\mathbf{u},\mathbf{p})(\delta\mathbf{u},\delta p) &= \int_\Omega ((\delta\mathbf{u}\cdot\nabla)\mathbf{u}).\mathbf{v} + ((\mathbf{u}.\nabla)\delta\mathbf{u}.\mathbf{v} + \nu\nabla\delta\mathbf{u} : \nabla\mathbf{v} \\ &\quad -\delta p\nabla\cdot\mathbf{v} - q\nabla\cdot\delta\mathbf{u} \end{aligned}$$

3.3.4 FreeFem++ Code for NSNewton.edp

Find the file NSNewton.edp in the "examples/chap3" directory and execute the code.

Only the Newton Raphson part of the code is reproduced here for a few comments.

color [hmargin=1.5cm,vmargin=1.5cm]geometry

```
// intial guess with B.C.
u1 = ( xˆ2+yˆ2) ¿ 2;
u2=0;

//  Physical parameter
real nu= 1./50, nufinal=1/200. ,cnu=0.5;

// stop test for Newton
real eps=1e-6;

verbosity=0;
while(1)  //  Loop on vicosity
{   int n;
```

```
real err=0;
for( n=0;n< 15;n++) // Newton Loop
{
solve Oseen([du1,du2,dp],[v1,v2,q]) =
int2d(Th) (  nu*(Grad(du1,du2)'*Grad(v1,v2) )
+ UgradV(du1,du2, u1, u2)'*[v1,v2]
+ UgradV( u1, u2,du1,du2)'*[v1,v2]
- div(du1,du2)*q - div(v1,v2)*dp
- 1e-8*dp*q // stabilization term
)
- int2d(Th) (  nu*(Grad(u1,u2)'*Grad(v1,v2) )
+ UgradV(u1,u2, u1, u2)'*[v1,v2]
- div(u1,u2)*q - div(v1,v2)*p
- 1e-8*p*q
)
+ on(1,du1=0,du2=0)
;
u1[] -= du1[];
u2[] -= du2[];
p[]  -= dp[];
real Lu1=u1[].linfty,  Lu2 = u2[].linfty , Lp = p[].linfty;
err= du1[].linfty/Lu1 + du2[].linfty/Lu2 + dp[].linfty/Lp;

cout << n << " err = " << err << " " << eps << " rey  =" << 1./nu << endl;
if(err < eps) break; // converge
if( n¿3 && err ¿ 10.) break; //  Blowup ????
}
```

- Note the Newton-Raphson iteration and update.
- *The boundary conditions are always homogenous*
- Note the use of *macros* to facilitate neat coding.
- Note the look break criterion to take care of blowup and convergence.

3.4 A Common trick for the Initial Guess

We note a "*common trick*" for providing an "*good*" initial guess.

Execute a transient analysis for large time and use the end answer as a guess for the steady state. We will see unsteady Navier-Stokes several times in this Workshop.

3.5 HomeWork

1. Derive the linearised form of the Navier-Stokes equation given above.(Note that this is very simple, one way to do it is by components).
2. Refer to Florian De Vuyst chapter on the *Stokes and Navier-Stokes* and understand the *convect* operator of FreeFem++ and also the introduction to *fractional step* methods given as a separate chapter in his manual.

4 Session 4: (a)The Convect Operator. (b) The Stokes equations via the Mixed Finite element Method

In this session, look at the convect *operator in FreeFem ++ via a simple example. The convect operator is used in the solution of the Navier Stokes equation.*

We solve the Stokes problem (Lid driven cavity example) via the mixed method. The linear algebra for the discretized equations is via the penalty method. The Uzawa method will be discussed in the next session.

We end with an example where the convection is driven by a Stokes flow.

§ The convect operator and an example.

§ The Stokes problem, example of a solution that uses the penalty method for the linear algebraic solution.

§ Plotting streamlines in 2-dimensional flows.

4.1 The convect operator

This is discussed in detail in the section on *color Convection* in Chapter 9 of the manual.

We summarise quickly as follows,

The aim is to solve the hyperbolic equation

$$\partial_t u(x,t) + \alpha(x,t) \cdot \nabla u(x,t) = f(x,t)$$

FreeFem++ implements the Characteristic-Galerkin method for convection operators. Suppose $X(t)$ is the trajectory of a particle. Then, along this trajectory we consider $u(t)$ by writing $u(X(t),t)$ and see that the governing equation to be solved is

$$\frac{du}{dt}(X(t),t) = f(X(t),t) \tag{4.1}$$

where the trajectory is governed by the ODE

$$\frac{dX}{dt}(t) = \alpha((X(t), t)$$

Please note that $X(t)$ and (X, t) are vectors. The one step backward convection by the method of Characteristics-Galerkin for the unknown $u(x, t)$ of our governing convection equation will be

$$\frac{1}{\tau}(u^{m+1}(x) - u^m(X^m(x))) = f^m(x)$$

where $X^m(x)$ is an approximation of the solution at $t = m\tau$ of the ordinary differential equation

$$\frac{dX}{dt}(t) = (X(t)) \text{ and the initial condition } X((m+1)\tau) = x$$

and $\alpha^m(x) = (\alpha_1(x, m\tau), \alpha_2(x, m\tau))$.

Exercise: Follow the details of the explanations in the manual where it is shown that the following approximation for the term $u^m(X^m(x))$ is valid.

$$u^m(X^m(x)) = u^m(x - \alpha^m\tau)$$

To summarise the operator *convect* is called as $convect([a_1, a_2], -dt, u0)$ gives out $u^m(X^m(x))$ by taking in the values of ^{m}, time step and $u^m(x)$.

Example: convect.edp: Study the example convect.edp in the manual. You will find the code in the *examples/tutorial* in the Sofware package.

This is the solution of the transport equation with an initial condition, and $f = 0$ and therefore $u^{m+1}(x, t) = u^m(X^m(x) = convect([a_1, a_2], -dt, u^m)$ and a_1, a_2 are given in the problem.

Note: mentioned in the manual: While the scheme is very powerful and unconditionally stable, it is not conservative and in rare cases it may also diverge.

4.2 2-D Stokes flow in a lid driven cavity

Our aim is to study the implementation of stokes flow in a lid driven cavity given in FreeFem++. Everything is neatly summarized in section 9.6 in the manual. We add a few notes to this excellent presentation in the manual,

- The weak form is given on page 264 just above equation (9.44). The abstract form of this weak form is given in equation 9.44 and 9.45. These have already been discussed in the class.

- The discretized weak form is given in 9.48 along with the conditions that ensure an unique solution. See Ern and Gurmond (Theory and Practice of finite elemenyts) that discusses a list of compatible finite elements. examples in the manual are Taylor Hood (P2/P1, P2 for velocity, P1 for pressure this is used in the example) and the *mini element* (P1b, P1, P1 bubble for velocity and P1 element for pressure, look up the example on page 52)

- The matrix form of the equations is given 9.50. This is discussed in detail in Ern and Guermond, Section 4.4 on the linear algebra associated with the matrix 9.50.

  We summarise the key points ,

  1. The matrix size is large. Direct methods are out of question. Most iterative methods require the matrix to be positive definite. The key observation is that 9.50 is symmetric but need not positive definite.

  2. A solution is the *Penalty method* which is given in equation 9.51. The system of matrices lead to easy elimination of the pressure (p_h) variable. It can be shown that the matrix shown in equation 9.52 i.e., $(A+(1/\epsilon)B^TB)$ is symmetric and positive definite and sparse and thus can be solved by a known technique.

  3. Further, a theorem that proves that the perturbed solution is not too far from the original one is given by Ern and Guermond.

- Another option is to use the Uzawa method which we will study in the next session via the example in the manual (StokesUzawa.edp).

4.3 Convection driven by Stokes flow

We use *convect* and *Stokes flow* to study an example of convection driven by Stokes flow. Please use the example *Convection.edp* on page 42 of the manual by Florian De Vuyst.

4.4 Plotting streamlines

Please try this nice exercise which has a *neat visual* end result.

Note however that the following is only true for 2-D. The idea is to take in the velocity answers and plot *streamlines* of the flow. Streamlines are visual lines of flow (i.e., the velocities are tangents to the stream lines). For, this you need to solve the following equation,

Find ψ such that

$$-\Delta\psi = \nabla \times u \quad \text{curl of u}$$

Use homogenous Dirichlet boundary conditions on the boundary. The solution i.e., lines of constant ψ will be the streamlines. Look up the simple proof for this so called stream function or prove it yourself in a few lines.

Code this in FreeFem++ and visualise your Stokes flow answers.

4.5 Homework

1. Go through the *varf* function usage in FreeFem++ manual Section colorblue 6.12. Read up the *Uzawa method* used in the Stokes problem without penalty.

2. Use the *convect* operator to write the weak form of the Navier-Stokes equation. For the answer, see the latter half of example *Cavity.edp* (geometry is 2-D lid driven cavity) in the FreeFem++ manual and also the example in Chapter 5 in the manual of Florian De Vuyst (geometry is 2-D flow over a cylinder).