Finite Element in Fluids

Assignment 1

Manisha Chetry

23/05/2016

Given a domain $\Omega = (0,2) \times (0,3) \in R^2$. The boundary Γ, with Dirichlet and Neumann boundary conditions such that $= \Gamma_d \cup \Gamma_n$, is defined by the following closed set as

$$\Gamma_1 = (0,0) \times (0,\frac{3}{2}); \Gamma_2 = (0,0) \times (\frac{3}{2},3); \Gamma_3 = (0,2) \times (3,3); \Gamma_4 = (2,2) \times (0,3); \Gamma_5 = (0,2) \times (0,0)$$

Exercise 1

The equation for 2D steady convection-diffusion reaction problem with the unknown , the convective term a, the reaction term σ and source term s is as follows:

$$a \cdot \nabla u - \nabla \cdot (\nu \nabla u) + \sigma u = s \quad in \quad \Omega$$
$$u = u_D \quad on \quad \Gamma_D$$
$$n \cdot \nu \nabla u = h \quad on \quad \Gamma_N$$

In order to obtain the weak form, PDE is multiplied by a weight function w such that $w = 0$ on Γ_D.Integrating over the computational domain Ω we get,

$$\int_\Omega w\,(a.\nabla u - \nabla.(\nu \nabla u) + \sigma u) d\Omega = \int_\Omega w s d\Omega \quad in \quad \Omega$$

Integrating by parts the diffusion term and applying divergence theorem we get,

$$\int w(a.\nabla u) d\Omega + \int \nabla w.(\nu \nabla u) d\Omega + \int w\sigma u d\Omega = \int w s d\Omega + \int_{\Gamma_n} w h d\Gamma \quad in \quad \Omega$$

Equation can be rewritten in compact form as:

$$a(w,u) + c(w,u,a) + (w,\sigma u) = (w,s) + (w,h)_{\Gamma_N} \tag{1}$$
$$a(w^h,u^h) + c(a;w^h,u^h) + \sigma(w^h,u^h) = (w^h,s) + (w^h,h)_{\Gamma_N}$$

Now Discretizing the weak form using Galerkin's method and approximation to the solution,

$$u^h(x) = \sum_j u_j N_j(x) w = N_i$$

We get,

$$a(N_i, N_j) + c(N_i, N_j, a) + \sigma(N_i, N_j) = (N_i, s) + (N_i, h)_{\Gamma_N}$$

After assembling of the elements globally, an algebraic system of equation is obtained.Th system of equation can be written in the form as:

$$(C + K + F)u = f$$

Where u is the vector of the unknown values. C, K and F are respectively the convective matrix, the diffusion matrix and the reaction matrix, while f is the contribution of the *source term*,and the prescribed flux h.

2b). Using Galerkin formulation, we solve the following four cases :

a.) Convection dominant case

$$a = 1, \quad \nu = 0.001, \quad \sigma = 0.001, \quad s = 0, \quad Pe = 100$$

b.)Reaction dominant case

$$a = 0.001, \quad \nu = 0.001, \quad \sigma = 1, \quad s = 0, \quad Pe = 0.1$$

c.)Convection dominant case considering the source term.

$$a = 1, \quad \nu = 0.001, \quad \sigma = 0, \quad s = 1, \quad Pe = 100$$

d.)Convection-reaction dominant case

$$a = 1, \quad \nu = 0.001, \quad \sigma = 1, \quad s = 0, \quad Pe = 100$$

Results

For element size h=0.2

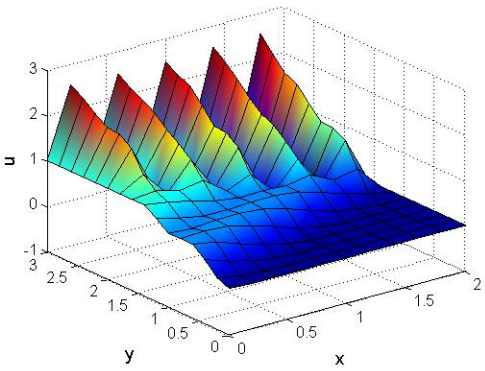

Figure 1: Galerkin for case 1

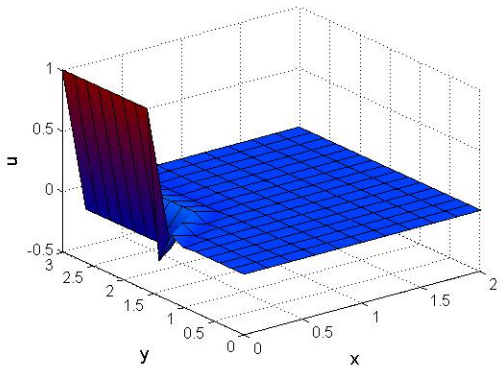

Figure 2: Galerkin for case 2

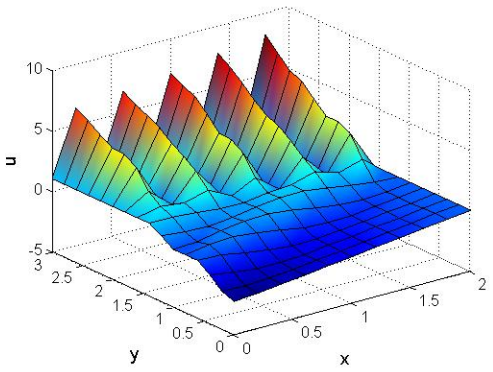

Figure 3: Galerkin for case 3

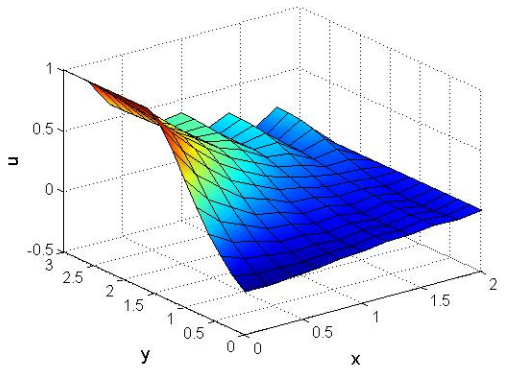

Figure 4: Galerkin for case 4

Observations: In the figure 1 we observe that Galerkin solutions give spurious oscillations in the nodes which is unacceptable. This is due to the presence of convective term which dominates diffusion and as a result of which Peclet number is very high. But in figure 2, it can be seen that node to node oscillations are greatly reduced and gives approximate solutions. This is due to reaction dominant case and very less Peclet number. In figure 3 and 4, non physical oscillations are reduced but the errors are still large.

In order to overcome the problem for a no acceptable solution, without modifying the Galerkin approach and without changing a,ν, σ and s, a mesh refinement is required. As the mesh element size h decreases, better results are expected than previous but still node to node oscillations are prevalent which is unacceptable.

For element size h=0.1

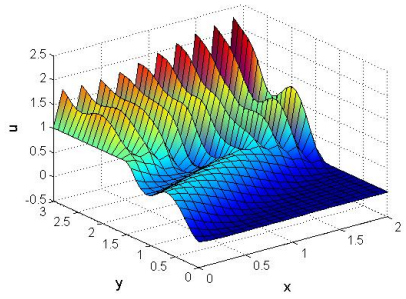

Figure 5: Galerkin for case 1

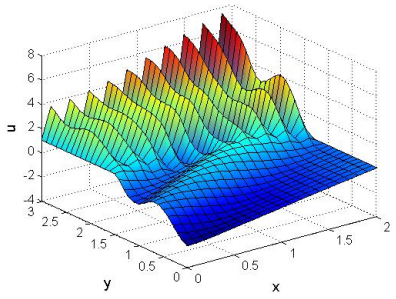

Figure 6: Galerkin for case 3

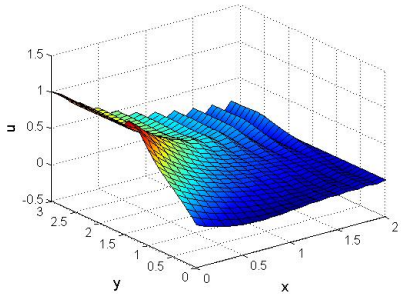

Figure 7: Galerkin for case 4

Another method was approached to overcome the problem, a modification of the Galerkin method was made.Galerkin method lacks diffusion so to overcome that, an artificial diffusion term $\overline{\nu}$ was added to the equation. The added numerical diffusion depends on the element size h and the parameters of the governing equation.

$$\overline{\nu} = \beta \frac{ah}{2} \quad with \quad \beta = cothPe - \frac{1}{Pe} \tag{2}$$

If artificial diffusion is added, the solution is smoothed in all directions but the formulation is not consistent. The results are shown below:

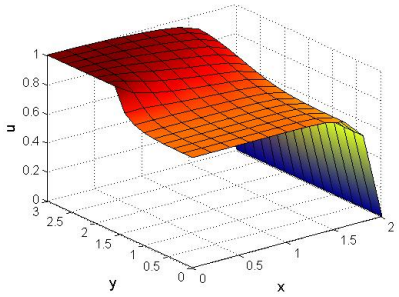

Figure 8: AD for case 1.

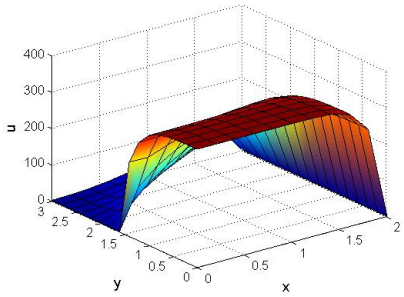

Figure 9: AD for case 3.

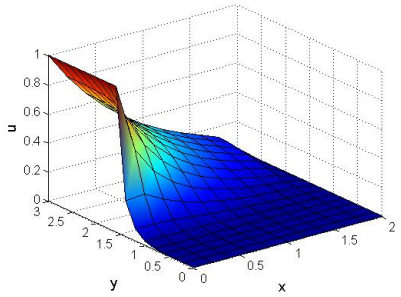

Figure 10: AD for case 4.

2c). Now SUPG and GLS methods are considered in order to solve the problem.SUPG and GLS implementations make sense when the Peclet number is larger than one and Galerkin cannot give stable results.In the given four cases in the assignment the second set of parameter would not give us any difference compared with the Galerkin method as peclet number is less than 1. So the rest of the cases (1,3 and 4) will give us better and stabilised results than galerkin. In order to appreciate the advantage of the stabilization techniques compared to Galerkin, the third case which is convection dominant without the reaction term is chosen.(Pe$\gg$ 1)

The formulation of these methods in 2D steady convection-diffusion-reaction problem are as follows:

$$a(w,u) + c(a;w,u) + (w,\sigma u) + \sum_e \int_{\Omega^e} P(w)\tau R(u)d\Omega = (w,s) + (w,h)_{\Gamma_N}$$

Where $P(w)$ depends on the method(SUPG/GLS), τ is the stabilization parameter and $R(u)$ is the residual. For SUPG method,

$$P(w) = a.\nabla w$$

For GLS:

$$P(w) = a\nabla w - \nabla(\nu\nabla w) + \sigma w$$

For linear elements $\nabla(\nu\nabla w) = 0$.

The stabilization term used for solving the problem is:

$$\tau = \frac{h}{2a}(1 + \frac{9}{Pe^2} + (\frac{\sigma h}{2a})^2)^{1/2}$$

The implementation of the SUPG method in matlab code:

$$Ke = Ke + (nu * (Nx' * Nx + Ny' * Ny) + N_ig' * (ax * Nx + ay * Ny) + N_ig' * sigma * (N_ig) +$$
$$tau * (ax * Nx + ay * Ny)' * ((ax * Nx + ay * Ny) + (\sigma * (N_ig)))) * dvolu;$$

$$fe = fe + (N_ig + tau * (ax * Nx + ay * Ny))' * (f_ig * dvolu);$$

The implementation of the GLS method in matlab code:

$$Ke = Ke + (nu * (Nx' * Nx + Ny' * Ny) + N_ig' * (ax * Nx + ay * Ny) + sigma * (N_ig' * N_ig) +$$
$$tau * (ax * Nx + ay * Ny + sigma * N_ig)' * (ax * Nx + ay * Ny + sigma * N_ig)) * dvolu;$$

$$fe = fe + (N_ig + tau * (ax * Nx + ay * Ny + sigma * N_ig))' * (f_ig * dvolu);$$

Results

For element size h=0.2

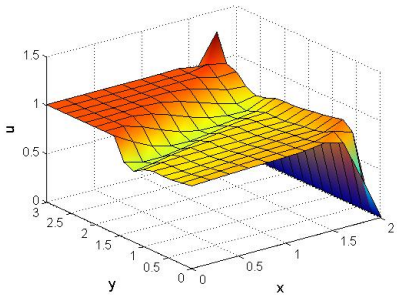

Figure 11: SUPG for case 1.

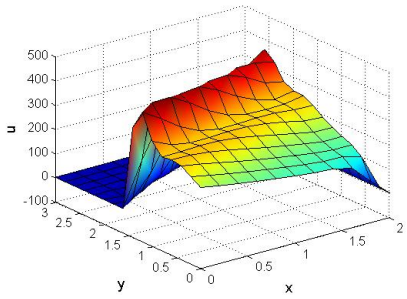

Figure 12: SUPG for case 3.

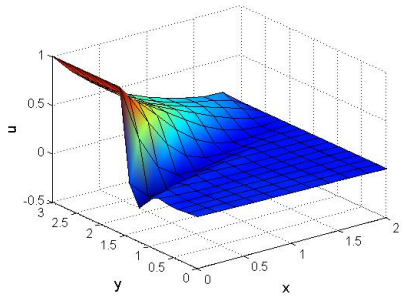

Figure 13: SUPG for case 4.

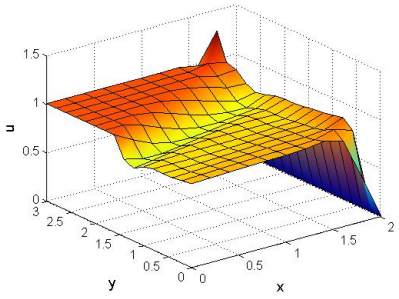

Figure 14: GLS for case 1

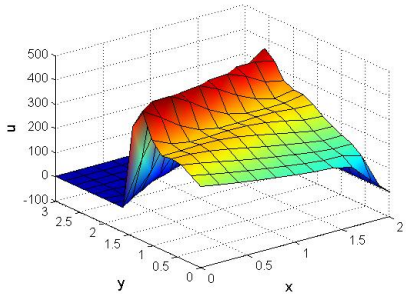

Figure 15: GLS for case 3

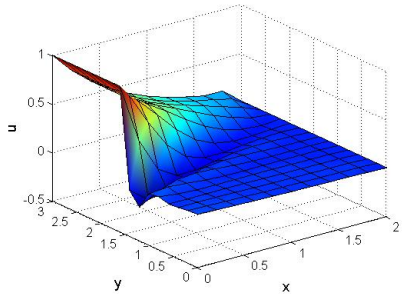

Figure 16: GLS for case 4

For element size h=0.1

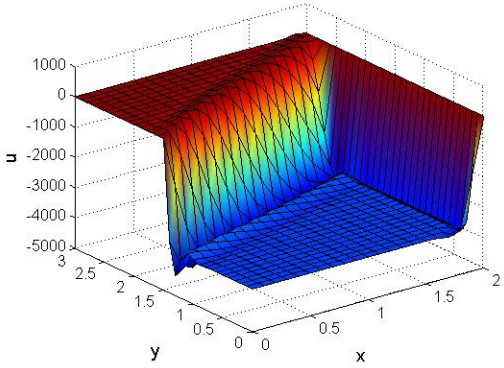

Figure 17: SUPG for case 3

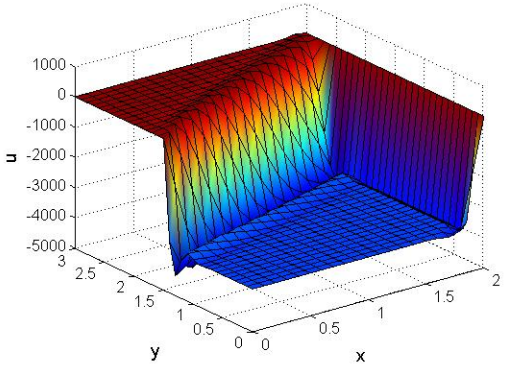

Figure 18: GLS for case 3

Observation We observe from the above figures that both SUPG and GLS gives similar results. But when the element size is reduced the solution changes drastically due to the presence of h in the stabilization parameter equation.Hence, more the mesh is refined the methods gives better results.

2d). Now the boundary conditions are changed to $u = 2$ in Γ_2 and $u = 1$in Γ_4.The BC in Γ_4 is modified in the code to impose Neumann BC so that the same solution is obtained. Following are the results obtained when Neumann boundary conditions are imposed vs Dirichlet boundry condition as imposed on the outlet boundary.

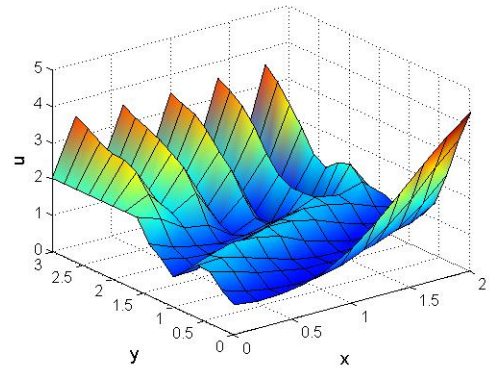

Figure 19: Galerkin with neumann boundary at outlet

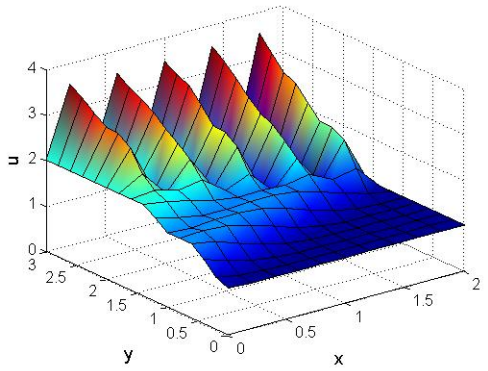

Figure 20: Galerkin with Dirichlet boundary at outlet

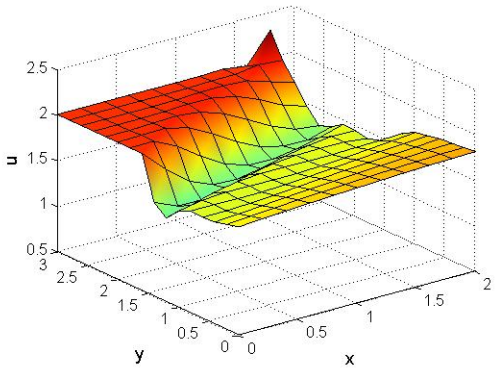

Figure 21: SUPG with neumann boundary at outlet

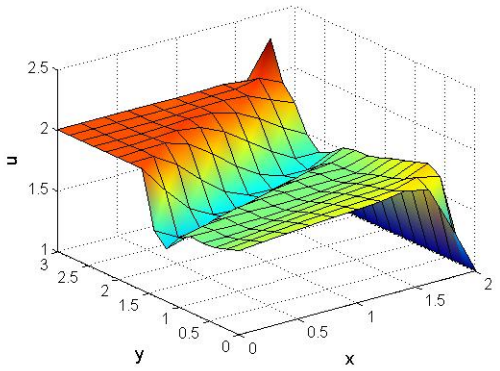

Figure 22: SUPG with Dirichlet boundary at outlet

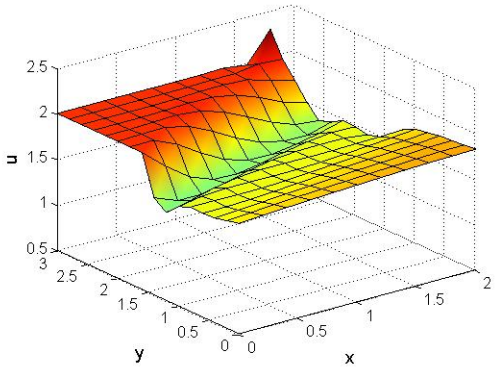

Figure 23: GLS with neumann boundary at outlet

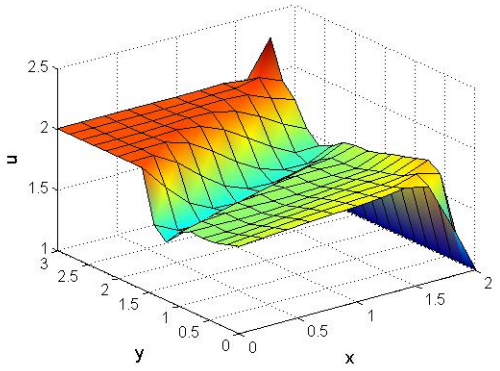

Figure 24: GLS with Dirichlet boundaryat outlet

Exercise 2

In this exercise the transient term is included in the 2D convection-diffusion-reaction PDE equation, the strong form is written as:

$$u_t + a \cdot \nabla u - \nabla \cdot (\nu \nabla u) + \sigma u = s \qquad in \quad \Omega \times]0, T[,$$
$$u(x, 0) = x(2 - x) \quad on \quad \Omega,$$
$$u = 1 on \quad \Gamma_2$$
$$u = 0 on \quad \Gamma_4$$
$$-\nu(n.\nabla)u = h \quad on \quad \Gamma_N$$

In order to discretize the problem above using the formulation for space among the ones described in Exercise 1, Galerkin spatial discretization is used.For time discretization two methods are used:
1. Crank Nicholson method
2.Two step third order TG method
3. Pade approximation of order R22

Crank-Nicolson +Galerkin Formulation:

The value of u^{n+1} is determined from u^n using the next relation:

$$\frac{u(t^{n+1}) - u(t^n)}{\Delta t} = \theta u_t(t^{n+1}) + (1 - \theta)u_t(t^n) + O(\frac{1}{2} - \theta)\Delta t, \Delta t^2)$$

Neglecting the truncation error and replacing u_t in the PDE we obtain:

$$\frac{\Delta u}{\Delta t} + \theta(a \cdot \nabla)\Delta u = \theta s^{n+1} + (1 - \theta)s^n - a \cdot \nabla u^n$$

For Crank-Nicolson method, $\theta = \frac{1}{2}$. This method is implicit, so a system of linear equation has to be solved. It is second order accuracy and it is unconditionally stable.

Multilpying with a test term and integrating by parts we get,

$$(w, \frac{\Delta u}{\Delta t}) - \theta(\nabla w, (a\Delta u) + \theta((a.n)w, \Delta u)_{\Gamma_{out}} = (\nabla w, au^n) - ((a.n)w, u^n)_{\Gamma^{out}} + (w, \theta h^{n+1} +$$
$$(1 - \theta)h^n)_{\Gamma_N^{in}} + (w, \theta h^{n+1} + (1 - \theta)h^n)_{\Gamma_N^{in}}$$

FE discretization:

In matrix form, it can be written as:

$$(\frac{1}{\Delta t}M - \theta C + \theta B^{out})\Delta u = f + (C - B^{out})u^n$$

where,

$$M_{ab} = \int_\omega N_a N_b d\Omega \quad Consistent \quad mass \quad matrix$$
$$C_{ab} = \int_\omega N_b(a.\nabla N_a)d\Omega \quad Convection \quad matrix$$
$$B_{ab}^{out} = \int_{\Gamma_{out}} N_b(a.n)d\Gamma \quad Outflow \quad Boundary \quad matrix$$

The implemented matlab code for Crank-Nicloson method is:

A = M + (dt/2)*C + (dt/2)*Mo;

B = - dt*C - dt*Mo;
f = dt*v1;

Where M and C are the mass matrix and the convective matrix. The Mo matrix is obtained by discretizing the term $a.n(w,u)$ on the outflow boundary and v1 is the vector related to source term.

Two step third order TG method (TG3-2S) It only involves 2nd order derivatives and achieves third order accuracy.

$$\widetilde{u}^n = u^n + \frac{1}{3}\Delta t u_t^n + \alpha \Delta t^2 u_{tt}^n$$

$$u^{n+1} = u^n + \Delta t u_t^n + \frac{1}{2}\Delta t^2 \widetilde{u}_{tt}^n$$

The parameter α influences the coefficient of the fourth order term in the overall time series. For a value of $\alpha = \frac{1}{9}$the TG3-2S method converts to TG3-1S.

Pade approximation of order R22

Pade approximation is fourth order time accurate discretization scheme:

$$u^{n+1} = (1 + \Delta t\frac{\partial}{\partial t} + \frac{1}{2!}\Delta t^2\frac{\partial^2}{\partial t^2} + ...)u^n = exp(\Delta t\frac{\partial}{\partial t})u^n$$

The Pade R22 is an implicit 4th order scheme which can be obtained in the following compact form:

$$\frac{\Delta u}{\Delta t} - W\Delta u_t = w u_t^n$$

where,

$$\Delta u = \begin{pmatrix} u^{n+\frac{1}{2}} - u^n \\ n^{n+1} - u^{n++\frac{1}{2}} \end{pmatrix}$$

$$W = \frac{1}{24}\begin{bmatrix} 7 & -1 \\ 13 & 5 \end{bmatrix}$$

$$w = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Results

The problem was solved using Crank Nicolson galerkin formulation:
For t=110

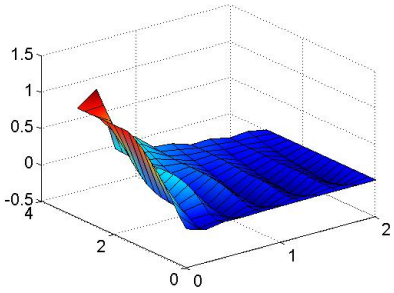

Figure 25: CN for case 1.

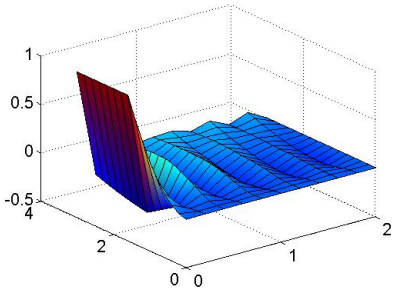

Figure 26: CN for case 2.

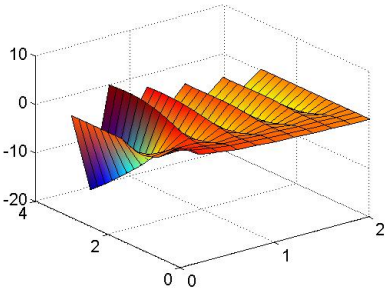

Figure 27: CN for case 3.

c). For time analysis the Courant's number has to be taken into account. Courant number is given by:

$$C = \frac{|a|\Delta t}{h}$$

So for different discretization time steps it was computed. Initially it was run for time step, t=100 and the results are shown above.But we observe that for time steps less than 110 and more than 110, the results don't vary much. But time to compute the problem decrease/increases. Table below shows how the time increases to compute as the number of time step increases. We observe that CN-Galerkin gives stable solutions as time step increases. From the figure below it can be observed that as time step increases, we get approximate solutions.

Time steps	50	120	160	220
Courant number	0.12566	0.069813	0.03927	0.02856
Comp.time	1.104	2.234	2.891	4.474

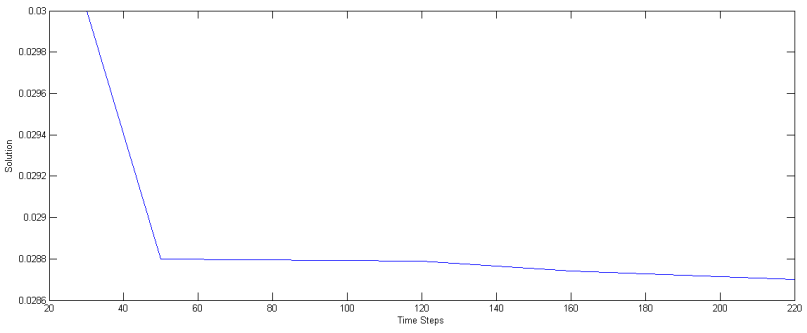

Figure 28: Computation at different time steps

Exercise 3

1). The strong form of Stokes equation for a steady case is given below:

$$-\nu\nabla^2 v + \nabla p = b \quad in \quad \Omega$$
$$\nabla \cdot v = 0 \quad in \quad \Omega$$
$$v = 0 \quad in \quad \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_5$$
$$v_y = -1 \quad in \Gamma_4$$

where ν is the kinematic viscosity and p is the kinematic pressure.
The weak form is obtained by integrating by parts and applying divergence theorem we get,

$$\int_\omega \nabla w \cdot \nu \nabla v d\Omega - \int_\omega p \nabla . w d\Omega = \int_\omega w.b d\Omega \quad \forall w \in \nu$$

$$\int_\omega q \nabla . v d\Omega = 0 \quad \forall q \in Q$$

Or in compact form,

$$a(w, v) + b(w, p) = (w, b)$$
$$b(v, q) = 0$$

$$\nu := H^1_{\Gamma_D}(\Omega) = w \in H^1(\Omega) | w = 0 \quad on \quad \Gamma_D$$

$$H^2_{\Gamma_D} = w \in H^2(\Omega) w = -1 \quad on \quad \Gamma_D$$

Approximating $v = v^h$ and pressure $p = p^h$,the Galerkin formulation of the Stokes problem is:

$$\begin{cases} a(w^h, v^h) + b(w^h, p^h) = (w^h, b^h) \\ b(v^h, q^h) = 0 \end{cases}$$

Finite dimensional Space

$$\begin{pmatrix} K & G \\ G^T & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ h \end{pmatrix}$$

Where K is the viscosity matrix, G is the discrete gradient operator and G^T is the discrete divergence operator.

Choosing element and mesh size is an important aspect for quality of the solution.For simulation, taylor Hood elements (Q2-Q1) elements have been used which is LBB stable. When we use a continuous space for pressure, then DOF for pressure can be saved a lot. Velocity space also becomes big enough to have the divergence stability which is one of the requisite condition.Well, for triangular bilinear elements also it gives similar results. But for linear elements there is spurious pressure nodes and oscillations in velocity field. So Q2-Q1 element is chosen which gives the best results out of rest of the elements(trianlge and triangle with bubble function). The results are shown below:

Q2-Q1 elements:

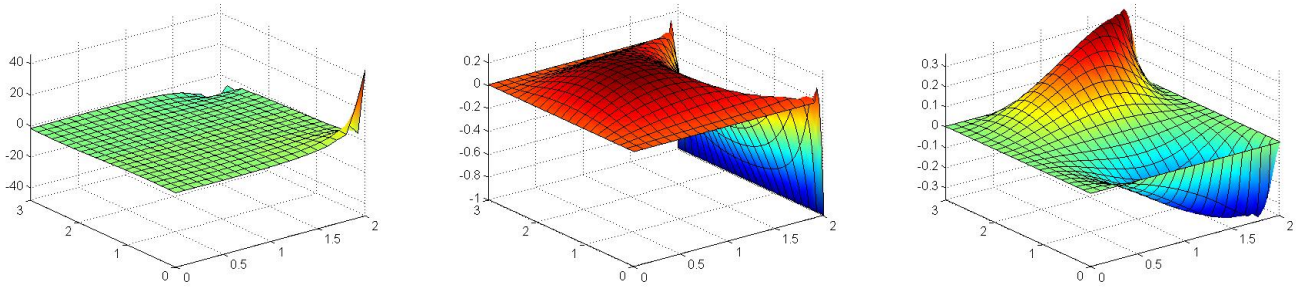

Figure 29: Pressure, velocity in y and x direction respectively for Q2-Q1 elements

Triangular bilinear elements

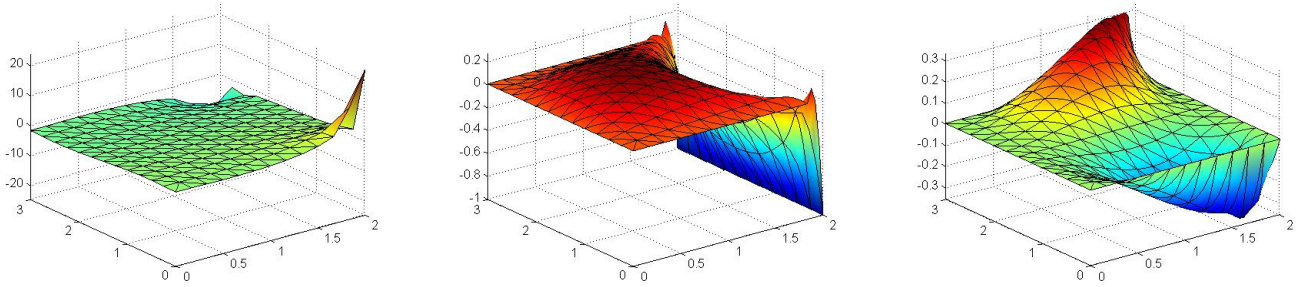

Figure 30: Pressure, velocity in y and x direction respectively for triangular elements

Q1-Q1(LBB not stable)

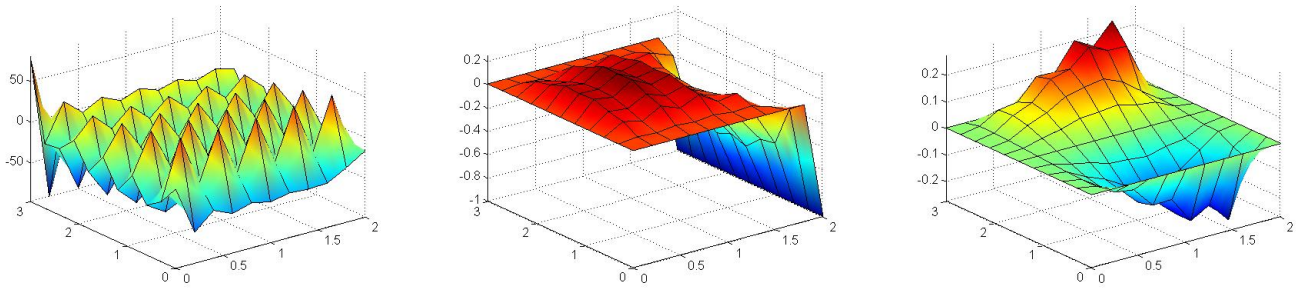

Figure 31: Pressure, velocity in y and x direction respectively for Q1-Q1 elements

2). Now navierstokes.m has been used to compute the solution of the Navier-Stokes equations using one specific spatial discretization and type of element for Re=1,100,1000 and 2000. The number of iterations needed for convergence is shown below in tabluar form:

For Re = 1:

Iteration	Velocity increment [m/s]
0	1
1	4.981520e-03
2	8.600491e-05
3	8.841132e-07

For Re = 100:

Iterations	Velocity increment [m/s]
0	1
1	3.255234e-01
2	1.110820e-01
...	...
11	=2.382659e-05

For Re = 1000:

iteration	Velocity increment [m/s]
0	1.153554e+00
1	1.269226e+00
2	6.933016e-01
...	...
99	1.010572e+01

For Re = 2000:

iteration	Velocity increment [m/s]
0	1.565798e+00
1	1.747213e+00
2	6.161955e+00
...	...
99	1.713691e+02

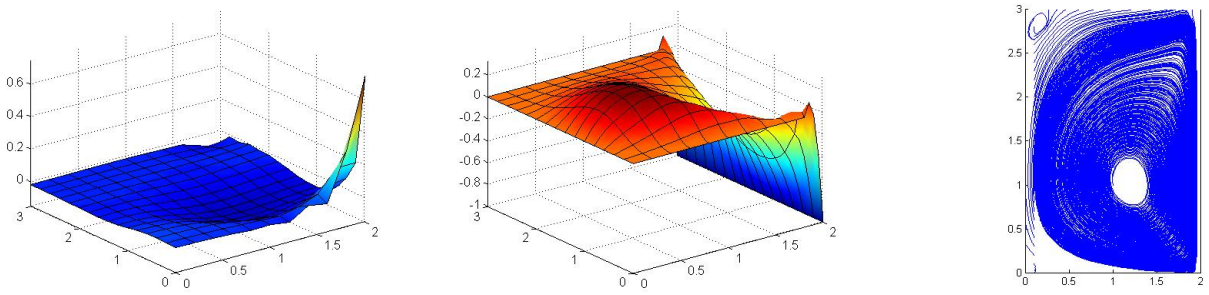

Figure 32: Diagram showing pressure, velocity field and streamlines for Re=100

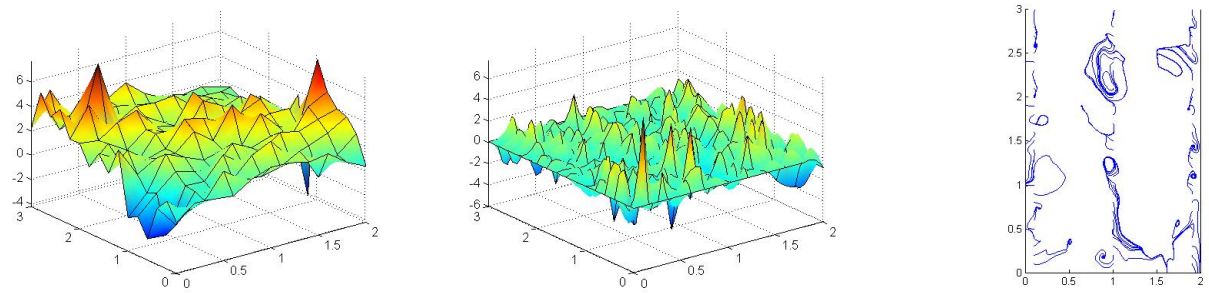

Figure 33: Diagram showing pressure, velocity field and streamlines for Re=1000

Appendices

Modification in **main.m** *file for exercise 1*

```
% This program solves a convection-diffusion reaction problem
% in a  domain [0,2]x[0,3]

clear; close all; clc


disp(' ')
disp('This program solves a convection-diffusion reaction equation on [0,2]x[0,3]')
disp(' ')
%disp('No source term is considered');

% PDE coefficients
a=cinput('Norm of Convective term',1);
disp('Convection velocity is');
velo = [a*cos(pi/6),a*sin(pi/6)];
disp(velo);
nu = cinput('Diffusion coefficient', 0.001);
disp(' ')
```

```
% nodes on which solution is u=1
nodesDir1 = nodes_x0( X(nodes_x0,2) >=1.5);
% nodes on which solution is u=0
nodesDir0 = [ nodes_x1 ];

% nodes on which solution is u=2
%nodesDir2 = nodes_x0( X(nodes_x0,2) >=1.5);
% nodes on which solution is u=1
%nodesDir1 = [ nodes_x1 ];
% Boundary condition matrix
C = [nodesDir1, ones(length(nodesDir1),1);
     nodesDir0, zeros(length(nodesDir0),1)];
```

```
%Neumann BC on nodes_x1
%C = [nodesDir2, 2*ones(length(nodesDir2),1)];
%N=nodesDir1;
%known_flux = [  -0.0002
 %  -0.0009
  %0.0010
   %0.0007
   %-0.0002
   %-0.0090
   %-0.0396
   %-0.0797
   %-0.1001
   %-0.1060
   %-0.1246
   %-0.1497
   %-0.1659
   %-0.1882
   %-0.10301;
  %-0.1246
  %-0.1497
  %-0.1659
  %-0.1882
  %-0.1030];

%f(N)=f(N)+known_flux;

nDir = size(C,1);
neq  = size(f,1);
A = zeros(nDir,neq);
A(:,C(:,1)) = eye(nDir);
b = C(:,2);
```

Modification in **FEMsystem.m** file in EX.1

```
    % GLS method
    Ke = Ke + (nu*(Nx'*Nx+Ny'*Ny) + N_ig'*(ax*Nx+ay*Ny) +sigma*(N_ig'*N_ig)+...
        tau*(ax*Nx+ay*Ny+sigma*N_ig)'*(ax*Nx+ay*Ny+sigma*N_ig))*dvolu;
    aux = N_ig*Xe;
    f_ig = SourceTerm(aux);
    fe = fe + (N_ig+tau*(ax*Nx+ay*Ny+sigma*N_ig))'*(f_ig*dvolu);

else
    % SUPG
    Ke = Ke + (nu*(Nx'*Nx+Ny'*Ny) + N_ig'*(ax*Nx+ay*Ny) +sigma*(N_ig'*N_ig)+...
        tau*(ax*Nx+ay*Ny)'*(ax*Nx+ay*Ny+sigma*N_ig))*dvolu;
    aux = N_ig*Xe;
    f_ig = SourceTerm(aux);
    fe = fe + (N_ig+tau*(ax*Nx+ay*Ny))'*(f_ig*dvolu);
end
```

Modification in **Method.m** *file for exercise 2*

```
function [ A,B,methodName ] = Method(M,C,K,A,B,dt,method,sigma)

switch method
    case 1 % Crank-Nicolson + Galerkin

    A = M/dt + 0.5*(C+K+sigma*M);
    B = -(C+K+sigma*M);
    methodName = 'CN+Galerkin';

    case 2  % CN+SUPG
     A = M/dt + 0.5*(C+K+sigma*M)+A;
     B = -(C+K+sigma*M+B);
     methodName = 'CN+SUPG';
    case 3
     A = M/dt;
     B = C+K+sigma*M;
     methodName = 'TG 3S+Galerkin';
```

Modification in **Computevelocity.m** *file for exercise 2*

```
function Conv = ComputeVelocity(X,velo)
%
% Velocity at the mesh nodes

if velo == 1
    Conv = [-X(:,1), -X(:,2)];
elseif velo == 2
    n = size(X,1);
    Conv = [ones(n,1), zeros(n,1)];
elseif velo == 3
    Conv = zeros(size(X)) / sqrt(2);
else
    error('not available velocity')
end
```

Modification in **BoundaryConditions.m** *file for exercise 2*

```
function [A,b,nDir] = BoundaryConditions(X,velo)

x1 = min(X(:,1)); x2 = max(X(:,1)); xM = (x1+x2)/2;
y1 = min(X(:,2)); y2 = max(X(:,2)); yM = (y1+y2)/2;

if velo == 1
   % nodes_x1 = find( (abs(X(:,1) - x1)<1e-6)  ); %\Gamma_2
    nodes_x1 = find( (abs(X(:,1) - x1)<1e-6) & (X(:,2)>=yM) ); %\Gamma_2
    nodes_x2 = find( (abs(X(:,1) - x2)<1e-6)  );
    %nodes_y1 = find( (abs(X(:,2) - y1)<1e-6) & (X(:,1)>=xM) );
    %nodes_y2 = find( (abs(X(:,2) - y2)<1e-6) & (X(:,1)<=xM) );
    % nodesDir = unique([nodes_x1; nodes_x2; nodes_y1; nodes_y2]);
elseif velo == 2
    nodesDir = find( (abs(X(:,1) - x2)<1e-6));
elseif velo == 3
    nodes_x2 = find( (abs(X(:,1) - x2)<1e-6));
    nodes_y2 = find( (abs(X(:,2) - y2)<1e-6));
    nodesDir = unique([nodes_x2;nodes_y2]);
end
```

Modification in **Stokessystem.m** *file for exercise 3*

```
Ke = Ke + mu*(Nx'*Nx+Ny'*Ny)*dvolu;
Ge = Ge - NP_ig'*dN*dvolu;
x_ig = N_ig(1:ngeom)*Xe;
f_igaus = SourceTerm(x_ig);
fe = fe + Ngp'*f_igaus*dvolu;
end
```

Modification in **BC.m** *file for exercise 3*

```
function [A, b, nDir, confined] = BC(X,dom,n)
% [A, b, nDir, confined] = BC(X,dom,n)
% Matrices to impose Dirichlet boundary conditions using Lagrange
% multipliers on a rectangular domain
% Input:
%    X: nodal coordinates
%    dom: domain description [x1,x2,y1,y2]
%    n: number of velocity degrees of freedom
% Output:
%    A,b: matrix and r.h.s. vector to impose the boundary conditions using
%         Lagrange multipliers
%    nDir: number of prescribed degrees of freedom
%    confined:

x1 = dom(1);  x2 = dom(2);
y1 = dom(3);  y2 = dom(4);

tol = 1e-6;
```

```
tol = 1e-6;
nodesX1 = find(abs(X(:,1)-x1) < tol);
nodesX2 = find(abs(X(:,1)-x2) < tol);
nodesY1 = find(abs(X(:,2)-y1) < tol & abs(X(:,1)-x2)>tol);
nodesY2 = find(abs(X(:,2)-y2) < tol & abs(X(:,1)-x2)>tol);


%confined flow(velocity is imposed for all the nodes in the boundary)
confined = 1;

nodesDir=unique([nodesX1;nodesY1;nodesY2]);
nodesDirg4=unique(nodesX2);

%no. of prescribed dof
nDir=2*length(nodesDir);
nDirg4=2*length(nodesDirg4);

%horizontal and vertical component
```

```
nodesDir=unique([nodesX1;nodesY1;nodesY2]);
nodesDirg4=unique(nodesX2);

%no. of prescribed dof
nDir=2*length(nodesDir);
nDirg4=2*length(nodesDirg4);

%horizontal and vertical component
C = [2*nodesDir-1; 2*nodesDir
    2*nodesDirg4-1; 2*nodesDirg4];

A = zeros(nDir+nDirg4,n);
A(:,C) = eye(nDir+nDirg4);

%imposed value(velocity is set to zero on the boundary)
b = [zeros(nDir,1);zeros(nDirg4/2,1);-ones(nDirg4/2,1)];

nDir=nDir+nDirg4;
end
```