Finite Elements in FLuids

Assignment 1

Arjun Ajay*

Master of Science in Computational Mechanics, UPC, Barcelona

E-mail: arjunajay100@gmail.com

Introduction

This assignment solves both the transient and steady convection diffusion reaction problem, with the unknown term "ρ", the convective term "$\mathbf{a}$", the reaction term "σ", and the source term "s". The domain $\Omega = (0, 2) \times (0, 3) \in \mathbb{R}^2$, its boundaries and the corresponding dirichlet boundary conditions are shown in the Figure. 1.

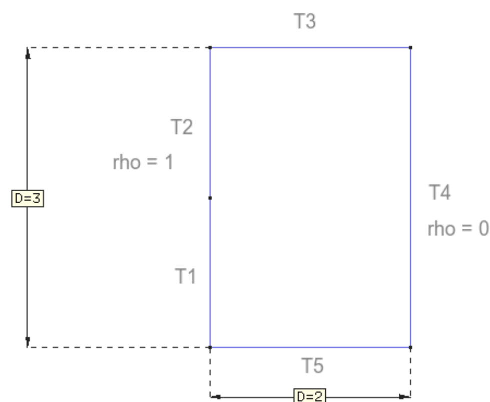

Figure 1: Domain and Boundary conditions

The unsteady convection diffusion reaction problem is given by eqn. 1:

$$\rho_t + \boldsymbol{a} \cdot \boldsymbol{\nabla}\rho - \nabla \cdot (\nu \boldsymbol{\nabla}\rho) + \sigma\rho = s \qquad in \;\; \Omega \tag{1}$$

$$\rho = 1 \qquad \text{in } \Gamma 2$$

$$\rho = 0 \qquad \text{in } \Gamma 4$$

For the space discretization, SUPG and GLS methods are used, while for the time discretization the implicit Pade approximations (R_{11} and R_{22}) and explicit Pade approximation (R_{20}) are used. The notations used in this assignment are from the book 'Finite element methods for flow problems, J.Donea, A.Huerta, 2003'.

a) Weak form of the Problem

Implicit Pade Approximation

The time discretization in compact form is:

$$\frac{\Delta\rho}{\Delta t} - \boldsymbol{W}\Delta\rho_t = \boldsymbol{w}\rho_t^n \tag{2}$$

where, $\Delta\rho_t$ and ρ_t^n can be written using eqn. 1 as :

$$\Delta\rho_t = \Delta s - \boldsymbol{a} \cdot \boldsymbol{\nabla}\Delta\rho + \nabla \cdot (\nu \boldsymbol{\nabla}\Delta\rho) - \sigma\Delta\rho$$

$$\rho_t^n = s^n - \boldsymbol{a} \cdot \boldsymbol{\nabla}\rho^n + \nabla \cdot (\nu \boldsymbol{\nabla}\rho^n) - \sigma\rho^n$$

To derive the weak form of the problem, we use a test function v. Therefore, eqn. 2 in weighted residual form can be written as :

$$(v, \frac{\Delta\rho}{\Delta t}) - (v, \boldsymbol{W}\Delta\rho_t) = (v, \boldsymbol{w}\rho_t^n) \tag{3}$$

Substituting $\Delta\rho_t$ and ρ_t^n into eqn. 3:

$$(v, \frac{\Delta\rho}{\Delta t}) + (v, \boldsymbol{W}\boldsymbol{a} \cdot \boldsymbol{\nabla}\Delta\rho) - (v, \boldsymbol{W}\nabla \cdot (\nu\boldsymbol{\nabla}\Delta\rho)) + (v, \sigma\Delta\rho) = (v, \boldsymbol{w}s^n) - (v, \boldsymbol{w}\boldsymbol{a} \cdot \boldsymbol{\nabla}\rho^n)$$
$$+(v, \boldsymbol{w}\nabla \cdot (\nu\boldsymbol{\nabla}\rho^n)) - (v, \boldsymbol{w}\sigma\rho^n) + (v, \boldsymbol{W}\Delta s) \tag{4}$$

Integration by parts of eqn. 4, gives the weak form of the problem as :

$$(v, \frac{\Delta\rho}{\Delta t}) + (v, \boldsymbol{W}\boldsymbol{a} \cdot \boldsymbol{\nabla}\Delta\rho) + (\nabla v, \boldsymbol{W}\nu\boldsymbol{\nabla}\Delta\rho) + (v, \sigma\Delta\rho) = (v, \boldsymbol{w}s^n + \boldsymbol{W}\Delta s) - \Big[(v, \boldsymbol{w}\boldsymbol{a} \cdot \boldsymbol{\nabla}\rho^n)$$
$$+(v, \boldsymbol{w}\nu\boldsymbol{\nabla}\rho^n) + (v, \boldsymbol{w}\sigma\rho^n)\Big] \tag{5}$$

where the boundary terms are absent due to the neumann boundary condition = 0.

To introduce the stabilization, the following terms are defined:

$$\mathscr{L}(\cdot) = \boldsymbol{a} \cdot \boldsymbol{\nabla}(\cdot) - \nabla(\nu\boldsymbol{\nabla}(\cdot)) + \sigma(\cdot)$$

and the Residual ($\mathscr{R}$) given by:

$$\mathscr{R}(\Delta u) = \frac{\Delta\rho}{\Delta t} + \boldsymbol{W}\mathscr{L}(\Delta u) - \boldsymbol{w}\Big[s^n - \mathscr{L}(u^n)\Big] - \boldsymbol{W}\Delta s$$

The stabilization term for each element is given by $(\boldsymbol{\tau}\mathscr{P}(v), \mathscr{R}(\Delta u))$.

The perturbation operator $\mathscr{P}$ is defined for each spacial discretization method.

- SUPG - $\mathscr{P}(v) = \boldsymbol{W}\boldsymbol{a} \cdot \boldsymbol{\nabla}v$
- GLS - $\mathscr{P}(v) = \dfrac{v}{\Delta t} + \boldsymbol{W}\mathscr{L}(v)$

and $\boldsymbol{\tau}$ is the intrinsic time scale parameter.

Adding the stabilization term to eqn. 5, the stabilized weak form of the problem is given by:

$$(v, \frac{\Delta \rho}{\Delta t}) + (v, \boldsymbol{W} \boldsymbol{a} \cdot \boldsymbol{\nabla} \Delta \rho) + (\nabla v, \boldsymbol{W} \nu \boldsymbol{\nabla} \Delta \rho) + (v, \sigma \Delta \rho) + \Sigma_e (\boldsymbol{\tau} \mathscr{P}(v), \mathscr{R}(\Delta u)) \Omega^e = (v, \boldsymbol{w} s^n + \boldsymbol{W} \Delta s) - \Big[(v, \boldsymbol{w} \boldsymbol{a} \cdot \boldsymbol{\nabla} \rho^n) + (v, \boldsymbol{w} \nu \boldsymbol{\nabla} \rho^n) + (v, \boldsymbol{w} \sigma \rho^n) \Big] \tag{6}$$

using approximation of ρ as $\rho^h(x) = \Sigma_j \rho_j N_j(x)$ and galerkin method($v = N_i$), the matrix form of eqn. 6 can be written as:

$$\Big[\frac{\boldsymbol{M}}{\Delta t} + \boldsymbol{W}(\boldsymbol{C} + \boldsymbol{K} + \sigma \boldsymbol{M}) + \boldsymbol{S_l} \Big] \boldsymbol{\Delta \rho} = \boldsymbol{s} + \boldsymbol{s_{st}} - \Big[\boldsymbol{w}(\boldsymbol{C} + \boldsymbol{K} + \sigma \boldsymbol{M}) + \boldsymbol{S_r} \Big] \rho^n \tag{7}$$

where the matrix elements are defined as:

Mass matrix - $M_{ij} = \int_\Omega N_i N_j d\Omega$

Convection matrix - $C_{ij} = \int_\Omega N_i (\boldsymbol{a} \cdot \nabla N_j) d\Omega$

Diffusion matrix - $K_{ij} = \int_\Omega \boldsymbol{\nabla} N_i \nu \boldsymbol{\nabla} N_j d\Omega$

Stabilization matrix left - $S_{lij} = \Sigma_e \int_{\Omega_e} \boldsymbol{\tau} \mathscr{P}(N_i) \boldsymbol{W} \mathscr{L}(N_j) d\Omega_e$

Stabilization matrix right - $S_{rij} = \Sigma_e \int_{\Omega_e} \boldsymbol{\tau} \mathscr{P}(N_i) \boldsymbol{w} \mathscr{L}(N_j) d\Omega_e$

and the vector elements are defined as:

source vector - $s_i = \int_\Omega N_i (\boldsymbol{w} s^n + \boldsymbol{W} \Delta s) d\Omega$

source stabilization vector - $s_{sti} = \Sigma_e \int_{\Omega_e} \boldsymbol{\tau} \mathscr{P}(N_i) \boldsymbol{w} s^n d\Omega_e$

Explicit Pade Approximation

For the R_{20} method, the time discretization is given by:

$$\rho^{n+1/2} = \rho^n + \frac{\Delta t}{2}\rho_t^n \tag{8}$$

$$\rho^{n+1} = \rho^n + \Delta t \rho_t^{n+1/2}$$

The weak form of the eqn. 8 is given by:

$$(v, \rho^{n+1/2}) = (v, \rho^n) + (v, \frac{\Delta t}{2}(s^n - \boldsymbol{a} \cdot \boldsymbol{\nabla}\rho^n - \sigma\rho^n) - (\nabla v, \nu \boldsymbol{\nabla}\rho^n) \tag{9}$$

$$(v, \rho^{n+1}) = (v, \rho^n) + (v, \Delta t(s^{n+1/2} - \boldsymbol{a} \cdot \boldsymbol{\nabla}\rho^{n+1/2} - \sigma\rho^{n+1/2}) - (\nabla v, \nu \boldsymbol{\nabla}\rho^{n+1/2})$$

The eqn. 9 in Matrix form including stabilization can be written as:

$$\big[\boldsymbol{S_l} + \boldsymbol{M}]\rho^{n+1/2} = \big[\boldsymbol{S_l} + \boldsymbol{M}]\rho^n + \frac{\Delta t}{2}\big[\boldsymbol{s} - \boldsymbol{C} - \boldsymbol{K} - \sigma\boldsymbol{M} - \boldsymbol{S_r}]\rho^n \tag{10}$$

$$\big[\boldsymbol{S_l} + \boldsymbol{M}]\rho^{n+1} = \big[\boldsymbol{S_l} + \boldsymbol{M}]\rho^n + \Delta t\big[\boldsymbol{s} - \boldsymbol{C} - \boldsymbol{K} - \sigma\boldsymbol{M} - \boldsymbol{S_r}]\rho^{n+1/2}$$

where

$$S_{lij} = \Sigma_e \int_{\Omega_e} \boldsymbol{\tau}\mathscr{P}(N_i)N_j d\Omega_e$$
$$S_{rij} = \Sigma_e \int_{\Omega_e} \boldsymbol{\tau}\mathscr{P}(N_i)\mathscr{L}(N_j) d\Omega_e$$

and the other matrix and vector elements are same as in the previous case.

b) Modifications to the MATLAB code

The definitions of the shape functions, quadrature, Mesh and connectivities are taken similar to the previous code for solving an Unsteady problem in 2D. However, the FEM matrices function used to find the Mass, Convection, Diffusion matrix has been edited to add the stabilization terms for the GLS and SUPG stabilization.

As the stabilization term, for quadratic elements has a second order term $\nabla \cdot (\nu \nabla \rho)$, a Jacobian matrix to convert the second order derivatives from the Natural coordinates to the cartesian coordinates is added to the code.

```
Laplacian = [(Nxi_ig*Xe(:,1))^2 (Nxi_ig*Xe(:,2))*(Nxi_ig*Xe(:,1)) N2xi_ig*Xe(:,1)...
             (Nxi_ig*Xe(:,1))*(Nxi_ig*Xe(:,2)) (Nxi_ig*Xe(:,2))^2 N2xi_ig*Xe(:,2);
        (Neta_ig*Xe(:,1))*(Nxi_ig*Xe(:,1)) (Neta_ig*Xe(:,2))*(Nxi_ig*Xe(:,1)) N2xieta_ig*Xe(:,1) (Neta_ig*Xe(:,1))*(Nxi_ig*Xe(:,2))...
        (Neta_ig*Xe(:,2))*(Nxi_ig*Xe(:,2)) N2xieta_ig*Xe(:,2)
        0 0 Nxi_ig*Xe(:,1) 0 0 Nxi_ig*(Xe(:,2))
        (Nxi_ig*Xe(:,1))*(Neta_ig*Xe(:,1)) (Nxi_ig*Xe(:,2))*(Neta_ig*Xe(:,1)) N2etaxi_ig*Xe(:,1) (Nxi_ig*Xe(:,1))*(Neta_ig*Xe(:,2))
        (Nxi_ig*Xe(:,2))*(Neta_ig*Xe(:,2)) N2etaxi_ig*Xe(:,2)
        (Neta_ig*Xe(:,1))^2 (Neta_ig*Xe(:,2))*(Neta_ig*Xe(:,1)) N2eta_ig*Xe(:,1) (Neta_ig*Xe(:,1))*(Neta_ig*Xe(:,2))
        (Neta_ig*Xe(:,2))^2 N2eta_ig*Xe(:,2);
        0 0 Neta_ig*Xe(:,1) 0 0 Neta_ig*(Xe(:,2))];          % To relate the second derivative of shape fn in
                                                               % the natural coordinates and the cartesian coordinates
```

Figure 2: Second order derivative Jacobian Matrix

New stabilization terms are defined for the R11, R22 and R20 pade approximations. Especially, for the R22 method as it is a two stage implicit method, the stabilization matrix is a square matrix of twice the size of the number of points in the mesh.

A new function to define the system for each of the time discretization methods called *system_assg1.m* is defined. This function groups the matrices on the left and right hand side of the equation, so that its easier to solve them during the time loop.

```matlab
function [ A,B,methodName,Cb ] = system_assg1(methodt,M,K,C,St_l,St_r,dt,assg1)


sigma = assg1.sigma;
switch methodt
    case 0 %pade R11
        W = 1/2;
        A = M/dt + W*(C+K+sigma*M)+St_l;
        B = -(C+K+sigma*M+St_r);
        methodName = 'R11';
        Cb = 0;
    case 1  %pade R22
        L =C+K+sigma*M;
        A = (1/24)*[M/dt+7*L  -1*L; 13*L  M/dt+5*L]+St_l;
        B_int =-(C+K+sigma*M);
        B = (1/2)*[B_int zeros(size(M,1));zeros(size(M,1)) B_int]-St_r;
        methodName = 'R22';
        Cb = 0;
    case 2 %pade R20

        A = M+St_l;
        B = M+St_l-(dt/2)*(C+K+sigma*M+St_r);
        Cb = St_l-(dt)*(C+K+sigma*M+St_r);
        methodName = 'R20';
end

end
```

Figure 3: System Function

c) Test case: Transient problem

The value of the stabilization constant for implicit Pade is found using:

$$\boldsymbol{\tau} = \left[\frac{\boldsymbol{W}^{-1}}{\Delta t} + \left(\frac{2a}{h} + \frac{4\nu}{h^2} + \sigma\right)\boldsymbol{I}\right]^{-T}\boldsymbol{W}^{-1}$$

where W $= \frac{1}{2}$ for the R_{11} and W $= \frac{1}{2}[5 \;\; 1; -13 \;\; 7]$ for R_{22} implicit method. However, for the above test case, using the R_{22} $\boldsymbol{\tau}$ gives erroneous result in GLS stabilization. Therefore, the same $\boldsymbol{\tau}$ for R_{11} is used for R_{22} multiplying it by $[1 \;\; 1; 1 \;\; 1]$ to make it a 2×2 matrix.

For the explicit pade method, the stabilization constant is defined as:

$$\boldsymbol{\tau} = \left(\frac{2a}{h} + \frac{4\nu}{h^2} + \sigma\right)^{-1}$$

These values are taken from the book 'Finite element methods for flow problems, J.Donea, A.Huerta, 2003'.

The initial condition consists of a Gaussian hill in the middle of the domain and a boundary condition compliant slope near the boundary Γ_2 as shown in the fig. 4

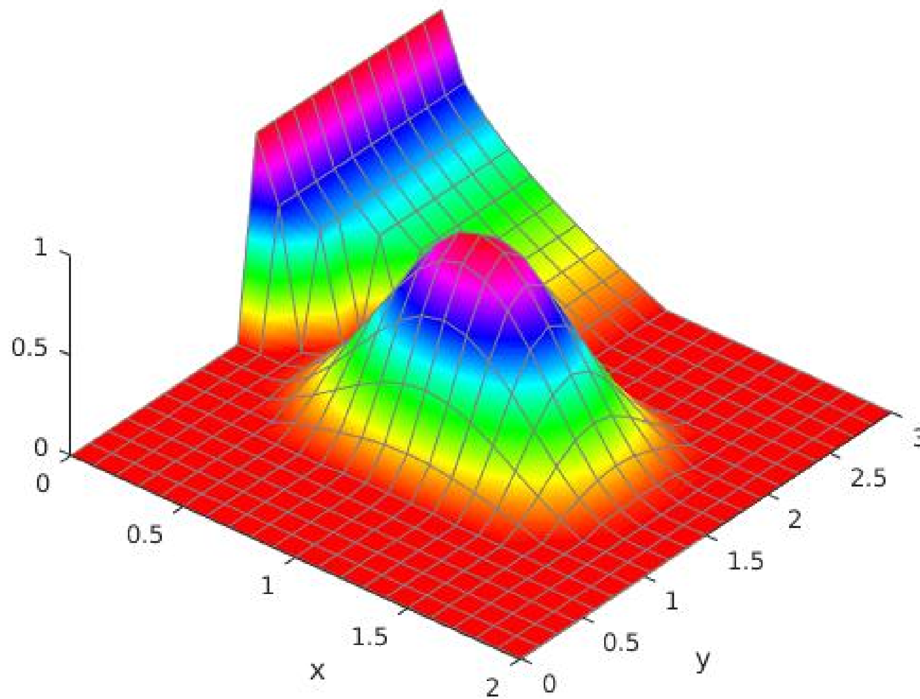

Figure 4: Initial Condition

Quadrilateral elements with linear interpolation are taken to show the results. Triangular elements and second order interpolation has also been implemented in the code. A 20×20 mesh is used for the simulation.

Test case 1

The Transient problem is solved with the following values for the convective velocity, diffusion parameter, reaction and source term:

$$a = (10^{-3}, 0),\ \nu = 10^{-3},\ \sigma = 1,\ s = 0$$

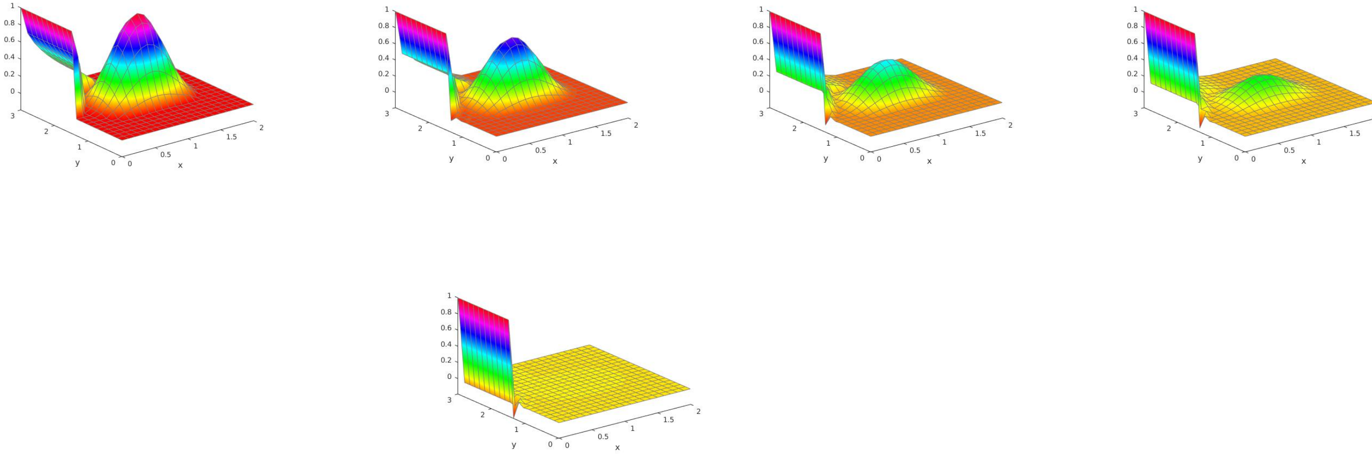

Figure 5: R_{11} with SUPG stabilization

R_{11} with SUPG stabilization was used for this simulation. As shown in the figure. 5 diffusion and reaction dominates convection for the test case and the Gaussian hill is diffused quickly with little convection. The boundary condition leads to boundary layer formation at Γ_2

Test case 2

To see the effect of convection a convection dominated flow with $a_x = 1$ and other parameters same is tested using R_{11} with GLS stabilization. The case chosen is :

$$a = (1, 0),\ \nu = 10^{-3},\ \sigma = 1,\ s = 0$$

The convection of the Gaussian hill is clearly seen in the Figure. 6. Oscillations are observed during the simulation as seen in the final sub figure, however, they were removed using a higher value of the parameter τ.

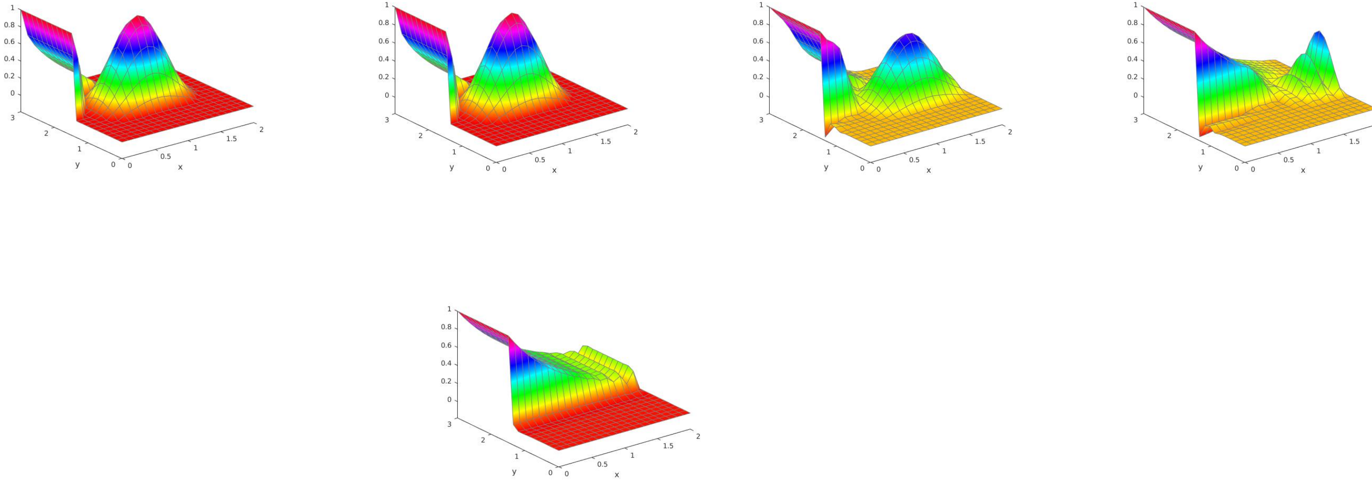

Figure 6: R_{11} with GLS stabilization

To compare the different methods of space and time discretization, the root mean square error in ρ with respect to the value of ρ from the R_{11} SUPG method as a function of time is plotted as shown in figure. 7 for the test case 1. R_{11} SUPG is chosen as the basis for comparison as the exact solution is not available and this is one of the easier methods to implement.

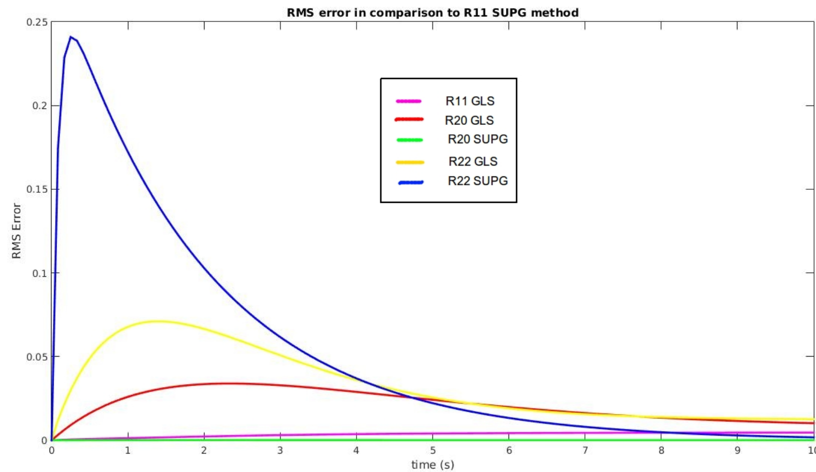

Figure 7: Error comparison with respect to R_{11} SUPG for Test case 1

As seen in the figure, all the methods converge to R_{11} SUPG solution as the time in-

creases. However the initial solutions from R_{22} for both SUPG and GLS have differences in comparison to R_{11} SUPG method. The SUPG stabilization method for R_{20} closely match the R_{11} SUPG solution.

d) Steady State Case

Three cases are studied to analyze the steady state behavior using both linear and quadratic elements. In all the figures for case 1 and 2 the range of ρ is from -0.5 to 1 in the z axis.

Case 1

$$a = (-1, 0),\ \nu = 10^{-3},\ \sigma = 10^{-3},\ s = 0$$

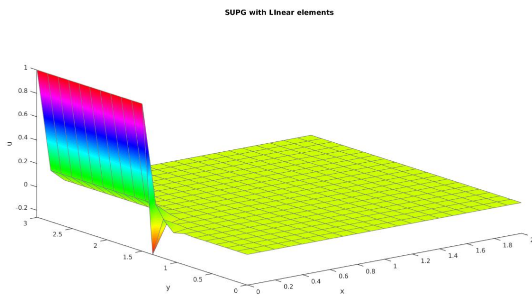

Figure 8: SUPG with linear elements

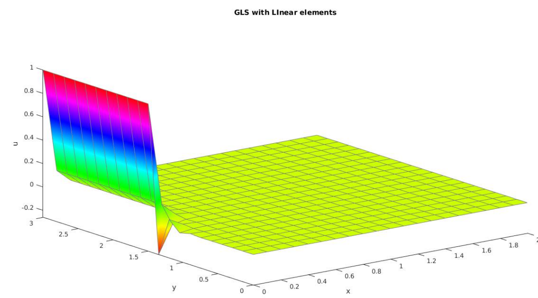

Figure 9: GLS with Linear elements

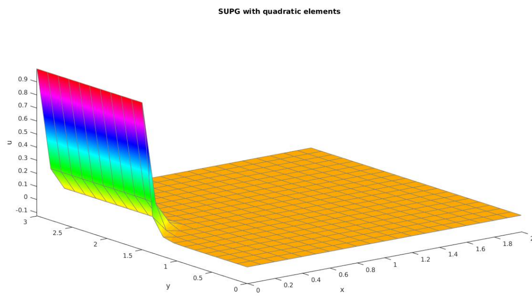

Figure 10: SUPG with Quadratic elements

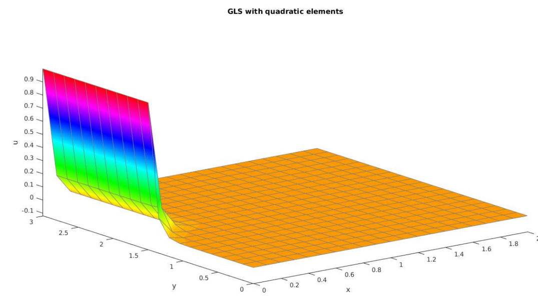

Figure 11: GLS with Quadratic elements

As seen in figures 8 and 9 as the convection velocity is high, convection dominates and at steady state there is a boundary layer formation near boundary Γ_2. Oscillation is observed in

the region of contact between Γ_2 and Γ_1 for the Linear elements. Using Quadratic elements, the oscillations are removed and results obtained are more accurate.

Case 2

$$a = (-10^{-3}, 0),\ \nu = 10^{-3},\ \sigma = 1,\ s = 0$$

As seen in figures 8 and 9 as the convection velocity is very small, reaction dominates and at steady state there is a boundary layer formation near boundary Γ_2.

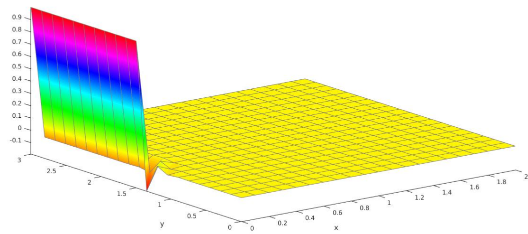

Figure 12: SUPG with linear elements

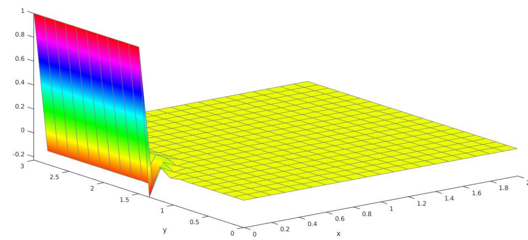

Figure 13: GLS with Linear elements

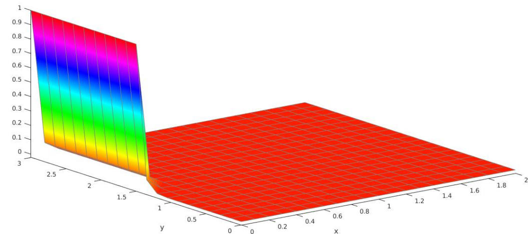

Figure 14: SUPG with Quadratic elements

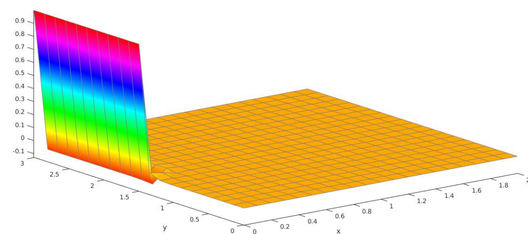

Figure 15: GLS with Quadratic elements

Oscillation is observed in the region of contact between Γ_2 and Γ_1 for the Linear elements. Using Quadratic elements, the oscillations are completely removed with SUPG stabilization.

In Case 2,except for at boundary Γ_2 where there is a boundary layer formation, the rest of the domain has ρ very close to 0, due to the high value of the reaction term.

Case 3

$$a = (-10^{-3}, 0),\ \nu = 10^{-3},\ \sigma = 0,\ s = 1$$

As seen in figures 8 and 9 as the convection velocity is very small and reaction is 0 there is a high value of ρ due to the constant source present through the domain. The variation at Γ_2 and Γ_4 are due to the boundary conditions imposed there, leading to boundary layer formation.

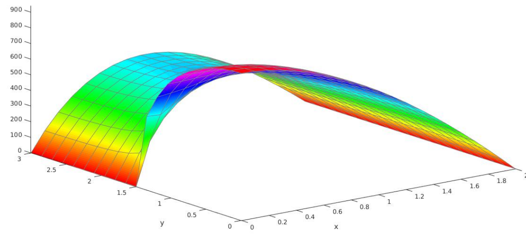

Figure 16: SUPG with linear elements

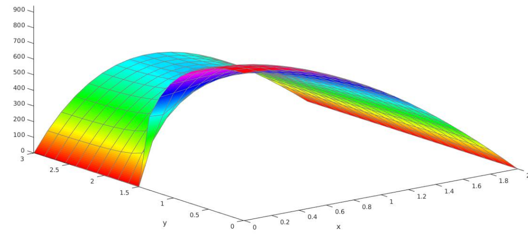

Figure 17: GLS with Linear elements

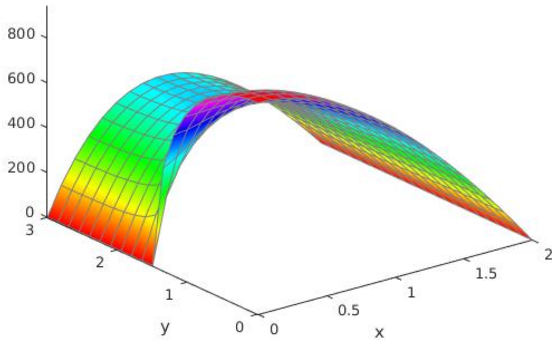

Figure 18: SUPG with Quadratic elements

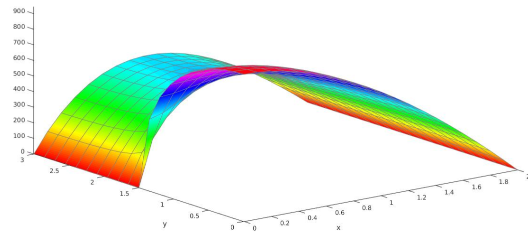

Figure 19: GLS with Quadratic elements

The range of ρ is from 0 to 800. All the methods give similar and accurate results.

Annex

The lists the Matlab code for the programs and functions used.

1. **main_assg1.m**

This is the main program from which other functions are called.

```
%this problem solves an unsteady convection diffusion reaction problem
%in a rectangular domain given by [0,2] x [0,3]

clear;close all;clc


disp('   ');
disp('This problem solves a convection diffusion reaction problem in the domain [0,2]x[0,3]');
disp('   ');
%disp('Both steady and unsteady problem can be solved');
disp('   ');
%disp('Source terms can be considered as required');

%PDE coefficients
ax = cinput_assg1('Enter the velocity in x direction ',1e-3);
disp('Convection velocity is ');
velo = [ax,0];
disp(velo);
a = norm(velo);
nu = cinput_assg1('Enter the Diffusion velocity ',1e-3);
disp(' ');
sigma = cinput_assg1('Enter the reaction ',1);
disp('  ');
dom = [0,2,0,3];

assg1.a = a;
assg1.nu = nu;
assg1.velo = velo;
assg1.dom = dom;
assg1.sigma = sigma;
%element type
elem = cinput_assg1('Enter the type of element- 0 - Quadrilateral | 1 - Triangular',0);  %
quadrilateral -> 0, triangle ->1
p =  cinput_assg1('Enter the type of element- 1 - Linear | 2 - Quadratic',1); % order of
interpolation

referenceElement = SetReferenceElement_assg1(elem,p);

nx = cinput_assg1('Input the number of elements in x direction (multiples of 5) ',20);
ny = cinput_assg1('Input the number of elements in y direction (multiples of 10)    ',20);
[X,T] = CreateMesh_assg1(dom,nx,ny,referenceElement);
%PlotMesh(T,X,'b-');

std = input('Do you want to solve a steady or unsteady problem? Enter 0 - steady | 1 -
unsteady    ');

disp('   ')
disp('The following stabilization methods are available for space discretization:');
disp('     [0] SUPG');
disp('     [1] GLS');
disp('   ');
```

```
methodsp = input('Enter the method for the space discretization:  ');
assg1.methodsp = methodsp;

npx = p*nx;
npy = p*ny;



if std == 1
    main_unsteady  %sub program dealing with the transient problem
else

   main_stead      %sub program dealing with the steady state problem

end
```

2. **main_unsteady.m**

This is the sub program which solves the transient problem.

```
disp('   ');
    tEnd = cinput_assg1('End time', 5);
    nStep = cinput_assg1('Number of time-steps', 120);
    dt = tEnd/nStep;


    disp('    ')
    disp('The following Pade methods are available for time discretization:   ');
    disp('     [0] R11 2nd order implicit scheme');
    disp('     [1] R22 4th order implicit scheme');
    disp('     [2] R20 2nd order explicit scheme');
    methodt = input('Enter the method for the time discretization    ');
    assg1.methodt = methodt;

    [ M,K,C,St_l,St_r ] = FEM_matrices_assg1( X,T,referenceElement,assg1,dt );
    %to find the finite element matrices the above function is used


    nPt = size(X,1);  %total number of points in the domain
    u = zeros(nPt,nStep); %initializing the unknown u to zero matrix with nPt rows and columns
=
    %the number of total time steps


    % BOUNDARY CONDITIONS
% Boundary conditions are imposed using Lagrange multipliers
nodes_y0 = [1:npx+1]';                          % Nodes on the boundary y=0
nodes_x1 = [2*(npx+1):npx+1:(npy+1)*(npx+1)]' ;     % Nodes on the boundary x=1
nodes_y1 = [npy*(npx+1)+npx:-1:npy*(npx+1)+1]' ;     % Nodes on the boundary y=1
nodes_x0 = [(npy-1)*(npx+1)+1:-(npx+1):npx+2]';     % Nodes on the boundary x=0

% nodes on which solution is u=1
nodesDir1 = [nodes_y1(end);nodes_x0( X(nodes_x0,2) > 1.5) ; nodes_x0( X(nodes_x0,2) ==
1.5)] ;

% nodes on which solution is u=0
nodesDir0 = [nodes_y0(end) ; nodes_x1];

bdof = [nodesDir1;nodesDir0];
ndofunk = setdiff(1:nPt,bdof);  %the unknown degree of freedoms
```

```
%initial conditions
u0    = InitialCondition(X);  %Function to set the initial condition for the transient case
%u1=zeros(nPt,1);
%u0(nodesDir1) = ones(size(nodesDir1));
u(:,1) = u0;

% Boundary condition matrix
Cbd = [nodesDir1,ones(length(nodesDir1),1);
     nodesDir0, zeros(length(nodesDir0),1)];
nDir = size(Cbd,1);
neq  = nPt;
A_lag = zeros(nDir,neq);
A_lag(:,Cbd(:,1)) = eye(nDir);    %Lagrange multiplier matrix
b_lag = Cbd(:,2);


[A,B,methodName,Cb] = system_assg1(methodt,M,K,C,St_l,St_r,dt,assg1); %function which
sets the A
% and B matrices for the equation Au(n+1) = B(u) depending on the type of
% method



if methodt == 0
      for n = 1:nStep
          Ktot = [A A_lag';A_lag zeros(nDir,nDir)];
          ftot = [B*u(:,n);b_lag];

          sol = Ktot\ftot;

          Du = sol(1:neq);
          u(ndofunk,n+1) = u(ndofunk,n) + Du(ndofunk); %updating the value of u for the new
timestep
          u(bdof,n+1) = u(bdof,n); %value of u for the boundary dof are not updated
      end
elseif methodt == 1
    for n = 1:nStep
        Alag = [A_lag zeros(size(A_lag));zeros(size(A_lag)) A_lag];
        Ktot = [A Alag';Alag zeros(2*nDir)]; % all the matrices have twice the size as the R22
        %method involves a 2 stage implicit method

        ftot = [B*[u(:,n);u(:,n)];b_lag;b_lag];

        sol = Ktot\ftot;
        Du1 = sol(1:neq); %Du1 is the change in u for first stage
        Du2 = sol(neq+1:2*neq);%change in u for the second stage
        Diff = Du1+Du2;% the net difference between u(n+1) and u(n)
        u(ndofunk,n+1) = u(ndofunk,n) + Diff(ndofunk);
        u(bdof,n+1) = u(bdof,n);
    end
elseif methodt == 2
    for n = 1:nStep
        Ktot = [A A_lag';A_lag zeros(nDir,nDir)];
        ftot_int = [B*u(:,n);b_lag];

        sol_int = Ktot\ftot_int; %explicit method
        u_int = sol_int(1:neq);



        ftot = [A*u(:,n)+Cb*u_int;b_lag];
        sol = Ktot\ftot;
        u(:,n+1) = sol(1:neq);
```

```
    end
end
disp(methodName);





% POSTPROCESS
% Plotting of the results

fid = figure;
% wObj = VideoWriter('out.avi');
% wObj.FrameRate = 25;
% open(wObj)

for i = 1:nStep
    figure(fid)
    clf
    patch('faces',T(:,1:4),'vertices',[X(:,1) X(:,2) u(:,i)],'facevertexcdata',u(:,i),...
          'FaceColor','interp','EdgeColor',[0.5,0.5,0.5]);
    colormap(hsv(100));
    zlim([min(min(u)) max(max(u))])
    xlabel('x')
    ylabel('y')
    view(3)
    pause(1/25)
    %     frame = getframe(gcf);
    %     writeVideo(wObj, frame);
end

% close(wObj);
```

3. **main_steady.m**

This is the sub program which solves the steady state.

```
    src = cinput_assg1('Enter the source ',0);
 [ K,f ] = FEM_matrices_steady( X,T,referenceElement,assg1,src );



    % BOUNDARY CONDITIONS
% Boundary conditions are imposed using Lagrange multipliers
nodes_y0 = [1:npx+1]';                        % Nodes on the boundary y=0
nodes_x1 = [2*(npx+1):npx+1:(npy+1)*(npx+1)]' ;     % Nodes on the boundary x=1
nodes_y1 = [npy*(npx+1)+npx:-1:npy*(npx+1)+1]' ;     % Nodes on the boundary y=1
nodes_x0 = [(npy-1)*(npx+1)+1:-(npx+1):npx+2]';     % Nodes on the boundary x=0

% nodes on which solution is u=1
nodesDir1 = [nodes_y1(end);nodes_x0( X(nodes_x0,2) > 1.5) ; nodes_x0( X(nodes_x0,2) ==
1.5)] ;

% nodes on which solution is u=0
nodesDir0 = [nodes_y0(end) ; nodes_x1];
% Boundary condition matrix
C = [nodesDir1, ones(length(nodesDir1),1);
     nodesDir0, zeros(length(nodesDir0),1)];
nDir = size(C,1);
neq  = size(f,1);
A = zeros(nDir,neq);
A(:,C(:,1)) = eye(nDir);
```

```
b = C(:,2);


% SOLUTION OF THE LINEAR SYSTEM
% Entire matrix
Ktot = [K A';A zeros(nDir,nDir)];
ftot = [f;b];

sol = Ktot\ftot;
Temp = sol(1:neq);
multip = sol(neq+1:end);


%POSTPROCESS



figure
patch('faces',T(:,1:4),'vertices',[X(:,1) X(:,2) Temp],'facevertexcdata',Temp,...
    'FaceColor','interp','EdgeColor',[0.5,0.5,0.5]);
colormap(hsv(100));
zlim([min(Temp) max(Temp)])
xlabel('x')
ylabel('y')
view(3)
```

4. **FEM_matrices_assg1.m**

This is the function which finds the FEM matrices for the transient problem.

```
function [ M,K,C,St_l,St_r ] = FEM_matrices_assg1( X,T,referenceElement,assg1,dt )
% [M,K,C] = FEM_system(X,T,referenceElement,assg1,dt)
% Matrix M,K,C obtained after discretizong a 2D unsteady convection-diffusion equation
%
% X:          nodal coordinates
% T:          connectivities (elements)
% referenceElement: reference element properties (quadrature, shape functions...)
% assg1: assg1 properties

velo = assg1.velo;
a = assg1.a;
nu = assg1.nu;
sigma = assg1.sigma;
methodsp = assg1.methodsp;
methodt = assg1.methodt;

Te = T(1,:); Xe = X(Te,:);
hx = max(Xe(:,1)) - min(Xe(:,1));
hy = max(Xe(:,2)) - min(Xe(:,2));
h = (hx+hy)/2;


nen = referenceElement.nen;
ngaus = referenceElement.ngaus;
wgp = referenceElement.GaussWeights;
N = referenceElement.N;
Nxi = referenceElement.Nxi;
Neta = referenceElement.Neta;
N2xi = referenceElement.N2xi;
N2eta = referenceElement.N2eta;
N2xieta = referenceElement.N2xieta;
```

```
N2etaxi = referenceElement.N2etaxi;

nElem = size(T,1);
nPt = size(X,1);

M = zeros(nPt);
K = zeros(nPt);
C = zeros(nPt);
if methodt == 0

   St_l = zeros(nPt); %Stabilization matrix on the left side of the equation
   St_r = zeros(nPt); %Stabilization matrix on the right side of the equation
   W = 1/2;
   w = 1;

   tau_p = 2/((2/dt)+(2*a/h)+(4*nu/(h^2))+sigma);
    disp(strcat('Recommended stabilization parameter = ',num2str(tau_p)));
    tau = cinput_assg1('Enter the Stabilization parameter',tau_p);
    if isempty(tau)
        tau = tau_p;
    end

elseif methodt == 1

     St_l = zeros(2*nPt);
     St_r = zeros(2*nPt);
     W = (1/24)*[7 -1;13 5];
     w = (1/2)*[1;1];

    tau_p= 2/((1/dt)+(2*a/h)+(4*nu/(h^2))+sigma);
    disp(strcat('Recommended stabilization parameter = ',num2str(tau_p)));
    tau = cinput_assg1('Enter the Stabilization parameter',tau_p);
    if isempty(tau)
         %tau = (inv(inv(W)/dt+((2*a/h)+(4*nu/(h^2))+sigma)*eye(2)))'*inv(W);
         tau = tau_p;
    end


else
    St_l = zeros(nPt);
    St_r = zeros(nPt);
   W = 1/2;
   w = 1;
   tau_p = 2/((2/dt)+(2*a/h)+(4*nu/(h^2))+sigma);
   %tau_p = 1/((2*a/h)+(4*nu/(h^2))+sigma);
    disp(strcat('Recommended stabilization parameter = ',num2str(tau_p)));
    tau = cinput_assg1('Enter the Stabilization parameter',tau_p);
    if isempty(tau)
        tau = tau_p;
    end
end




 % Loop on elements
for ielem=1:nElem
    Te = T(ielem,:);
    Xe = X(Te,:);
    [Me,Ke,Ce,St_le,St_re] =
EleMat(Xe,nen,ngaus,wgp,N,Nxi,Neta,N2xi,N2eta,N2xieta,N2etaxi,velo,nu,tau,methodsp,metho
dt,sigma,dt,W,w);
```

```
    % Assembly
    M(Te,Te) = M(Te,Te) + Me;
    K(Te,Te) = K(Te,Te) + Ke;
    C(Te,Te) = C(Te,Te) + Ce;

  if methodt == 0 || methodt == 2
     St_l(Te,Te) = St_l(Te,Te) + St_le;
     St_r(Te,Te) = St_r(Te,Te) + St_re;

  elseif methodt == 1
      Te = [T(ielem,:) T(ielem,:)+nPt];
      St_l(Te,Te) = St_l(Te,Te) + St_le;
      St_r(Te,Te) = St_r(Te,Te) + St_re;

  end
end

function [Me,Ke,Ce,St_le,St_re] =
EleMat(Xe,nen,ngaus,wgp,N,Nxi,Neta,N2xi,N2eta,N2xieta,N2etaxi,velo,nu,tau,methodsp,metho
dt,sigma,dt,W,w)
%

ax = velo(1);
ay = velo(2);

Me = zeros(nen);
Ke = zeros(nen);
Ce = zeros(nen);

if methodt == 0 || methodt == 2

   St_le = zeros(nen);
   St_re = zeros(nen);

elseif methodt == 1

    St_le = zeros(2*nen);
    St_re = zeros(2*nen);
end



for ig = 1:ngaus
    N_ig = N(ig,:);
    Nxi_ig = Nxi(ig,:);
    Neta_ig = Neta(ig,:);
    N2xi_ig = N2xi(ig,:);
    N2eta_ig = N2eta(ig,:);
    N2xieta_ig = N2xieta(ig,:);
    N2etaxi_ig = N2etaxi(ig,:);

    Jacob = [Nxi_ig*(Xe(:,1))   Nxi_ig*(Xe(:,2))       %for first order relation of
                                                  % derivatives in
                                                  % natural and
                                                  % cartesian
                                                  % coordinates
            Neta_ig*(Xe(:,1))  Neta_ig*(Xe(:,2))];

    Laplacian = [(Nxi_ig*Xe(:,1))^2 (Nxi_ig*Xe(:,2))*(Nxi_ig*Xe(:,1)) N2xi_ig*Xe(:,1)
(Nxi_ig*Xe(:,1))*(Nxi_ig*Xe(:,2)) (Nxi_ig*Xe(:,2))^2 N2xi_ig*Xe(:,2);
            (Neta_ig*Xe(:,1))*(Nxi_ig*Xe(:,1)) (Neta_ig*Xe(:,2))*(Nxi_ig*Xe(:,1)) N2xieta_ig*Xe(:,1)
(Neta_ig*Xe(:,1))*(Nxi_ig*Xe(:,2)) (Neta_ig*Xe(:,2))*(Nxi_ig*Xe(:,2)) N2xieta_ig*Xe(:,2)
```

```
           0 0 Nxi_ig*Xe(:,1) 0 0 Nxi_ig*(Xe(:,2))
           (Nxi_ig*Xe(:,1))*(Neta_ig*Xe(:,1)) (Nxi_ig*Xe(:,2))*(Neta_ig*Xe(:,1)) N2etaxi_ig*Xe(:,1)
(Nxi_ig*Xe(:,1))*(Neta_ig*Xe(:,2)) (Nxi_ig*Xe(:,2))*(Neta_ig*Xe(:,2)) N2etaxi_ig*Xe(:,2)
           (Neta_ig*Xe(:,1))^2 (Neta_ig*Xe(:,2))*(Neta_ig*Xe(:,1)) N2eta_ig*Xe(:,1)
(Neta_ig*Xe(:,1))*(Neta_ig*Xe(:,2)) (Neta_ig*Xe(:,2))^2 N2eta_ig*Xe(:,2);
           0 0 Neta_ig*Xe(:,1) 0 0 Neta_ig*(Xe(:,2))];          % To relate the second derivative of
shape fn in
                                                              % the natural coordinattes to the cartesian
coordinates

   dvolu = wgp(ig)*det(Jacob);

   res   = Laplacian\[N2xi_ig;N2xieta_ig;Nxi_ig;N2etaxi_ig;N2eta_ig;Neta_ig];
   Nx = res(3,:);
   Ny = res(6,:);
   Nxx = res(1,:);
   Nyy = res(5,:);
   Lp = Nxx+Nyy;  %the diffusion term which will be used for second order stabilization

   %source term not added at present
   Me = Me + N_ig'*N_ig*dvolu;
   aGradN = ax*Nx + ay*Ny;
   Ke = Ke + nu*(Nx'*Nx+Ny'*Ny)*dvolu;
   Ce = Ce + N_ig'*aGradN*dvolu;

   if methodt == 0

      if methodsp == 0  %SUPG
        St_le = tau*W*aGradN'*((N_ig/dt)+(W)*(aGradN+nu*Lp+sigma*N_ig))*dvolu;
        St_re = tau*W*aGradN'*(w*(aGradN+nu*Lp+sigma*N_ig))*dvolu;


      elseif methodsp == 1 %GLS
        St_le = tau*((N_ig/dt)+(W)*(aGradN+nu*Lp+sigma*N_ig))'*((N_ig/dt)+
(W)*(aGradN+nu*Lp+sigma*N_ig))*dvolu;
        St_re = tau*((N_ig/dt)+
(W)*(aGradN+nu*Lp+sigma*N_ig))'*(w*(aGradN+nu*Lp+sigma*N_ig))*dvolu;
      end

   elseif methodt == 1
      W3 = tau*w;
      W1 = W*W;
      W2 = W*w;
      if methodsp == 0  %SUPG
       St_re = [W2(1)*aGradN'*(aGradN+sigma*N_ig) zeros(nen) ; zeros(nen)
W2(2)*aGradN'*(aGradN+sigma*N_ig)]*dvolu;

        St_le = tau*[W(1,1)*aGradN'*(N_ig/dt)
+W1(1,1)*aGradN'*(aGradN+nu*Lp+sigma*N_ig)...
                  W(1,2)*aGradN'*(N_ig/dt)+W1(1,2)*aGradN'*(aGradN+nu*Lp+sigma*N_ig);

                W(2,1)*aGradN'*(N_ig/dt)+W1(2,1)*aGradN'*(aGradN+nu*Lp+sigma*N_ig)...
                W(2,2)*aGradN'*(N_ig/dt)+W1(2,2)*aGradN'*(aGradN+nu*Lp+sigma*N_ig)]*dvolu;

        St_re =tau*[W2(1)*aGradN'*(aGradN+nu*Lp+sigma*N_ig) zeros(nen) ; zeros(nen)
W2(2)*aGradN'*(aGradN+nu*Lp+sigma*N_ig)]*dvolu;


      elseif methodsp == 1 %GLS
```

```
        St_le = tau*[(N_ig/dt)'*(N_ig/dt) + W(1,1)*((N_ig/dt)'*(aGradN+nu*Lp+sigma*N_ig)+
(aGradN+nu*Lp+sigma*N_ig)'*(N_ig/dt))
+W1(1,1)*(aGradN+nu*Lp+sigma*N_ig)'*(aGradN+nu*Lp+sigma*N_ig)...
                        (N_ig/dt)'*(N_ig/dt) + W(1,2)*((N_ig/dt)'*(aGradN+nu*Lp+sigma*N_ig)+
(aGradN+nu*Lp+sigma*N_ig)'*(N_ig/dt))
+W1(1,2)*(aGradN+nu*Lp+sigma*N_ig)'*(aGradN+nu*Lp+sigma*N_ig);

                        (N_ig/dt)'*(N_ig/dt) + W(2,1)*((N_ig/dt)'*(aGradN+nu*Lp+sigma*N_ig)+
(aGradN+nu*Lp+sigma*N_ig)'*(N_ig/dt))
+W1(2,1)*(aGradN+nu*Lp+sigma*N_ig)'*(aGradN+nu*Lp+sigma*N_ig)...
                       (N_ig/dt)'*(N_ig/dt) + W(2,2)*((N_ig/dt)'*(aGradN+nu*Lp+sigma*N_ig)+
(aGradN+nu*Lp+sigma*N_ig)'*(N_ig/dt))
+W1(2,2)*(aGradN+nu*Lp+sigma*N_ig)'*(aGradN+nu*Lp+sigma*N_ig)]*dvolu;

           St_re =tau*[(N_ig/dt)'*(aGradN+nu*Lp+sigma*N_ig)+
W2(1)*(aGradN+nu*Lp+sigma*N_ig)'*(aGradN+nu*Lp+sigma*N_ig) zeros(nen) ; zeros(nen)
(N_ig/dt)'*(aGradN+nu*Lp+sigma*N_ig)+
W2(2)*(aGradN+nu*Lp+sigma*N_ig)'*(aGradN+nu*Lp+sigma*N_ig)]*dvolu;

 end

    elseif methodt == 2
        if methodsp == 0  %SUPG
          St_le = tau*aGradN'*(N_ig)*dvolu;
          St_re = tau*aGradN'*dt*(aGradN+nu*Lp+sigma*N_ig)*dvolu;


       elseif methodsp == 1 %GLS
          St_le = tau*((aGradN+nu*Lp+sigma*N_ig)'*N_ig)*dvolu;
          St_re = tau*dt*((aGradN+nu*Lp+sigma*N_ig)'*(aGradN+nu*Lp+sigma*N_ig))*dvolu;
       end

    end
    end
```

5. **systems_assg1.m**

This function groups the FEM matrices for each method together.

```
function [ A,B,methodName,Cb ] = system_assg1(methodt,M,K,C,St_l,St_r,dt,assg1)


sigma = assg1.sigma;
switch methodt
    case 0 %pade R11
        W = 1/2;
        A = M/dt + W*(C+K+sigma*M)+St_l;
        B = -(C+K+sigma*M+St_r);
        methodName = 'R11';
        Cb = 0;
    case 1  %pade R22
        L =C+K+sigma*M;
        A = (1/24)*[M/dt+7*L  -1*L; 13*L  M/dt+5*L]+St_l;
        B_int =-(C+K+sigma*M);
        B = (1/2)*[B_int zeros(size(M,1));zeros(size(M,1)) B_int]-St_r;
        methodName = 'R22';
        Cb = 0;
    case 2 %pade R20

        A = M+St_l;
        B = M+St_l-(dt/2)*(C+K+sigma*M+St_r);
```

```
        Cb = St_l-(dt)*(C+K+sigma*M+St_r);
        methodName = 'R20';
end

end
```

6. **FEM_matrices_steady.m**

This function finds the FEM matrices for the steady case.

```
function [ K,f] = FEM_matrices_steady( X,T,referenceElement,assg1,src )



velo = assg1.velo;
a = assg1.a;
nu = assg1.nu;
sigma = assg1.sigma;
methodsp = assg1.methodsp;
sr = src;

Te = T(1,:); Xe = X(Te,:);
hx = max(Xe(:,1)) - min(Xe(:,1));
hy = max(Xe(:,2)) - min(Xe(:,2));
h = (hx+hy)/2;


%ax = velo(1);
%ay = velo(2);


nen = referenceElement.nen;
ngaus = referenceElement.ngaus;
wgp = referenceElement.GaussWeights;
N = referenceElement.N;
Nxi = referenceElement.Nxi;
Neta = referenceElement.Neta;
N2xi = referenceElement.N2xi;
N2eta = referenceElement.N2eta;
N2xieta = referenceElement.N2xieta;
N2etaxi = referenceElement.N2etaxi;

nElem = size(T,1);
nPt = size(X,1);


K = zeros(nPt);
f = zeros(nPt,1);



   Pe = a*h/(2*nu);
   tau_p = h*(1 + 9/Pe^2)^(-1/2)/(2*a);
    disp(strcat('Recommended stabilization parameter = ',num2str(tau_p)));
    tau = cinput_assg1('Enter the Stabilization parameter',tau_p);
    if isempty(tau)
        tau = tau_p;
    end

 % Loop on elements
for ielem=1:nElem
```

```
    Te = T(ielem,:);
    Xe = X(Te,:);
    [Ke,fe] =
EleMat(Xe,nen,ngaus,wgp,N,Nxi,Neta,N2xi,N2eta,N2xieta,N2etaxi,velo,nu,tau,methodsp,sigma,
sr);
    % Assembly

    K(Te,Te) = K(Te,Te) + Ke;
    f(Te) = f(Te) + fe;

end


function [Ke,fe] =
EleMat(Xe,nen,ngaus,wgp,N,Nxi,Neta,N2xi,N2eta,N2xieta,N2etaxi,velo,nu,tau,methodsp,sigma,
sr)
%

ax = velo(1);
ay = velo(2);


Ke = zeros(nen);
fe = zeros(nen,1);

for ig = 1:ngaus
    N_ig = N(ig,:);
    Nxi_ig = Nxi(ig,:);
    Neta_ig = Neta(ig,:);
    N2xi_ig = N2xi(ig,:);
    N2eta_ig = N2eta(ig,:);
    N2xieta_ig = N2xieta(ig,:);
    N2etaxi_ig = N2etaxi(ig,:);

    Jacob = [Nxi_ig*(Xe(:,1))   Nxi_ig*(Xe(:,2))
             Neta_ig*(Xe(:,1))  Neta_ig*(Xe(:,2))];

    Laplacian = [(Nxi_ig*Xe(:,1))^2 (Nxi_ig*Xe(:,2))*(Nxi_ig*Xe(:,1)) N2xi_ig*Xe(:,1)
(Nxi_ig*Xe(:,1))*(Nxi_ig*Xe(:,2)) (Nxi_ig*Xe(:,2))^2 N2xi_ig*Xe(:,2);
             (Neta_ig*Xe(:,1))*(Nxi_ig*Xe(:,1)) (Neta_ig*Xe(:,2))*(Nxi_ig*Xe(:,1)) N2xieta_ig*Xe(:,1)
(Neta_ig*Xe(:,1))*(Nxi_ig*Xe(:,2)) (Neta_ig*Xe(:,2))*(Nxi_ig*Xe(:,2)) N2xieta_ig*Xe(:,2)
             0 0 Nxi_ig*Xe(:,1) 0 0 Nxi_ig*(Xe(:,2))
             (Nxi_ig*Xe(:,1))*(Neta_ig*Xe(:,1)) (Nxi_ig*Xe(:,2))*(Neta_ig*Xe(:,1)) N2etaxi_ig*Xe(:,1)
(Nxi_ig*Xe(:,1))*(Neta_ig*Xe(:,2)) (Nxi_ig*Xe(:,2))*(Neta_ig*Xe(:,2)) N2etaxi_ig*Xe(:,2)
             (Neta_ig*Xe(:,1))^2 (Neta_ig*Xe(:,2))*(Neta_ig*Xe(:,1)) N2eta_ig*Xe(:,1)
(Neta_ig*Xe(:,1))*(Neta_ig*Xe(:,2)) (Neta_ig*Xe(:,2))^2 N2eta_ig*Xe(:,2);
             0 0 Neta_ig*Xe(:,1) 0 0 Neta_ig*(Xe(:,2))];

    dvolu = wgp(ig)*det(Jacob);

    res   = Laplacian\[N2xi_ig;N2xieta_ig;Nxi_ig;N2etaxi_ig;N2eta_ig;Neta_ig];
    Nx = res(3,:);
    Ny = res(6,:);
    Nxx = res(1,:);
    Nyy = res(5,:);
    Lp = Nxx+Nyy;


      if methodsp == 0  %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+nu*Lp+sigma*N_ig))*dvolu;
         aux = N_ig*Xe;
```

```
        f_ig = SourceTerm(aux,sr);
        fe = fe + (N_ig+tau*(ax*Nx+ay*Ny))'*(f_ig*dvolu);


      elseif methodsp == 1 %GLS
        Ke = Ke + (nu*(Nx'*Nx+Ny'*Ny) + N_ig'*(ax*Nx+ay*Ny) + ...
         sigma*(N_ig'*N_ig)
+tau*(ax*Nx+ay*Ny+nu*Lp+sigma*N_ig)'*(ax*Nx+ay*Ny+nu*Lp+sigma*N_ig))*dvolu;
        aux = N_ig*Xe;
        f_ig = SourceTerm(aux,sr);
        fe = fe + (N_ig+tau*(ax*Nx+ay*Ny+nu*Lp+sigma*N_ig))'*(f_ig*dvolu);
      end


end
end
end
```

7. **InitialCondition.m**

This function defines the Initial condition for the transient problem

```
function u0 = InitialCondition(X)

nPt = size(X,1);
u0 = zeros(nPt,1);

% u0(X(:,1) <= 1.0) = 1;

sigma = 0.8; xref = 1; yref = 1.5;
xdim = ( X(:,1) - xref) / sigma;
ydim = ( X(:,2) - yref) / sigma;
ind = find( (xdim.^2 + ydim.^2)<=1 );
ind1 = find( X(:,1)<=1 & X(:,2)>=1.5);
u0(ind1) = 1-(sqrt(X(ind1,1)));
u0(ind) = 0.25*(1 + cos(pi*xdim(ind))).*(1 + cos(pi*ydim(ind)));
```