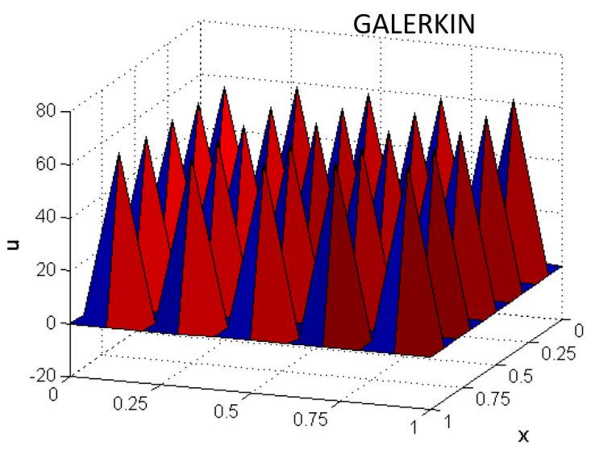
A. 2D Steady Transport

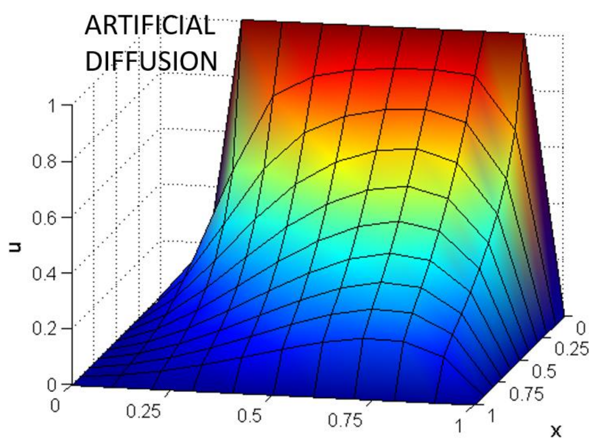
Goal:

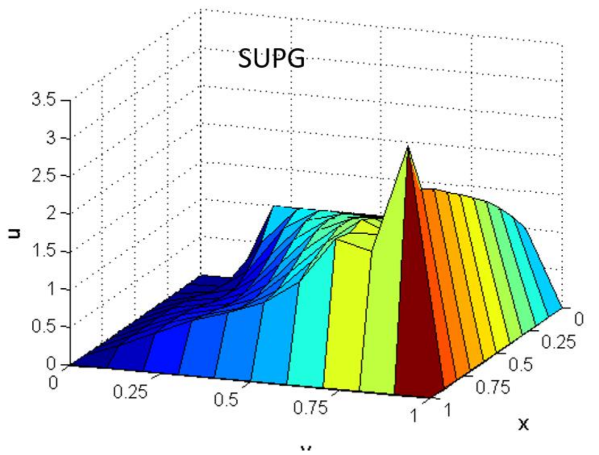
Using formulation such as Galerkin, streamline upwind (SU), Streamline Upward Petrov-Galerkin (SPUG) and Galerkin Least Square (GLS) method, to solve the defined problem.

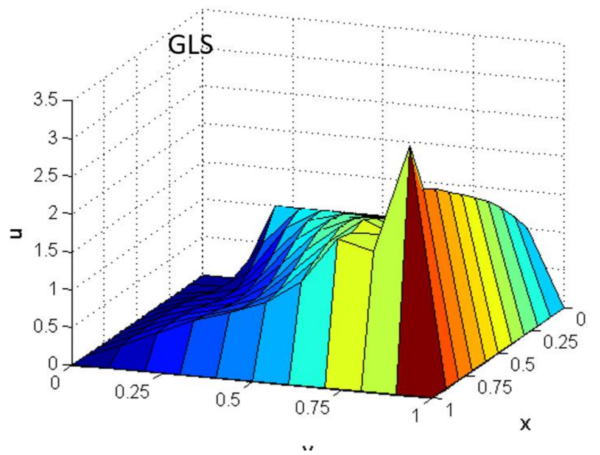
Graphs:

a. Neumann Boundary Condition (σ=0) The Galerkin results are wildly oscillatory and bear no resemblance to the exact results. Better results are obtained with the stabilized formulations. The arti_cial di_usion technique introduces excessive

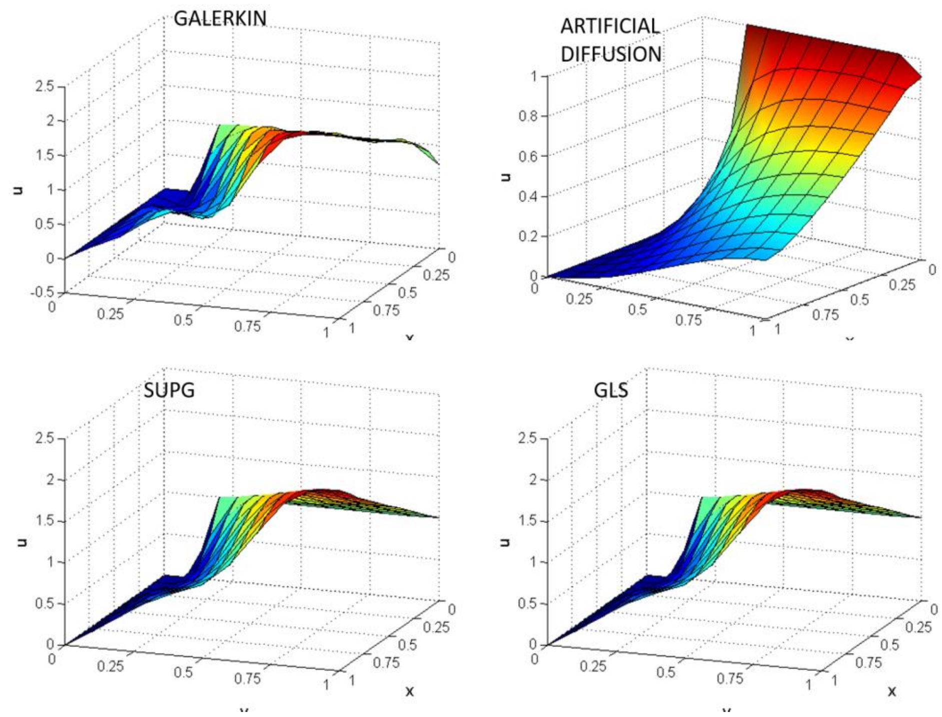






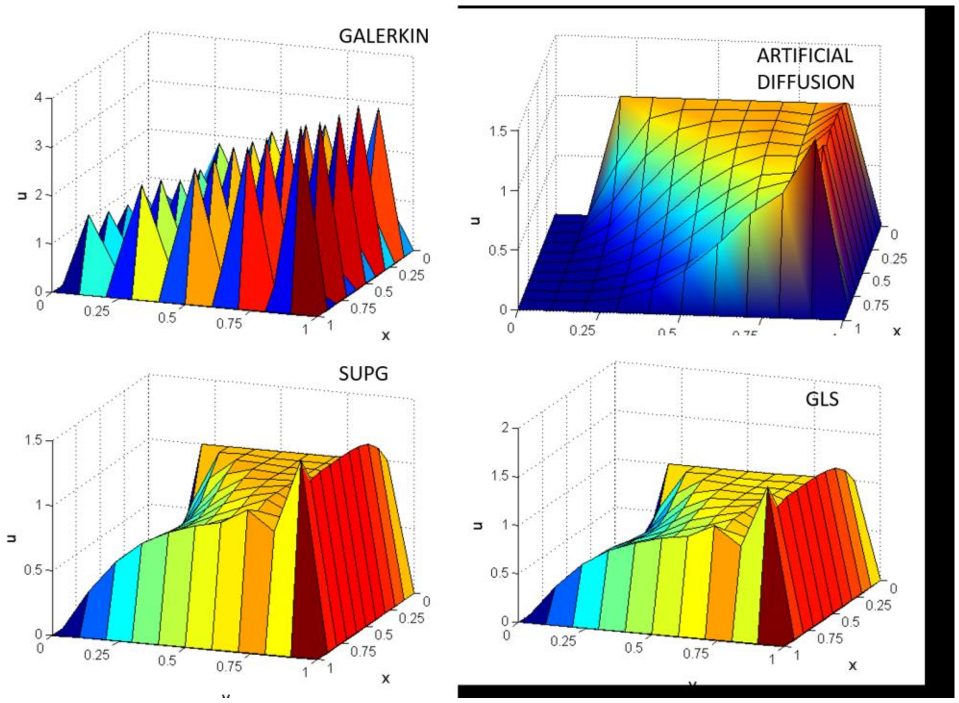


b. Dirichlet Boundary Conditions ($\sigma=0$) of pure convection. The Galerkin method is not able to satisfactorily resolve the discontinuity and produces spurious oscillations. The artificial diffusion method and the SUPG method yield better results, but SUPG introduces less crosswind diffusion.



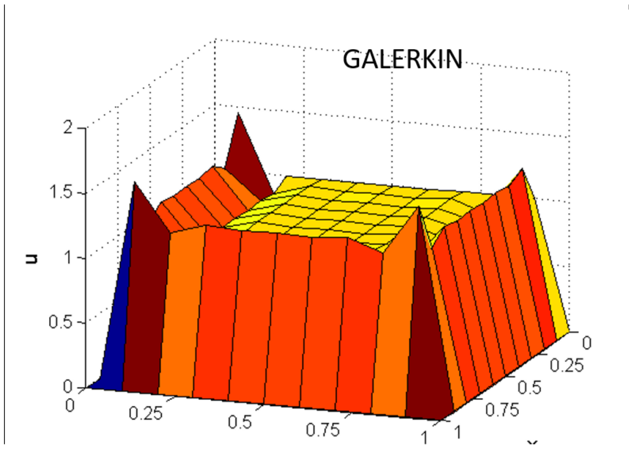
c. Convection-Reaction Dominated Case

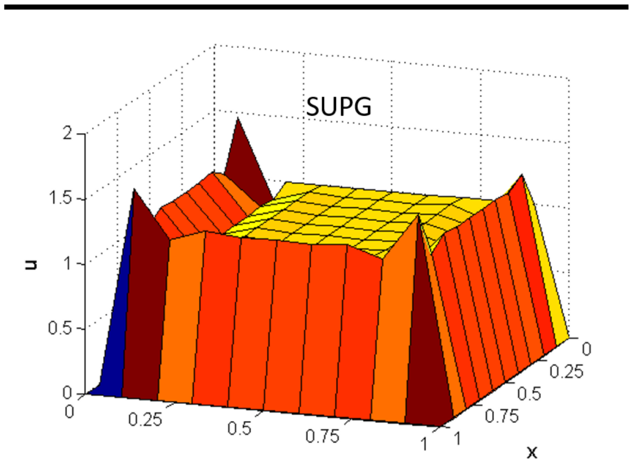
Dirichlet Boundary Conditions ($||a||=1/2$, $\nu=10^{-4}$, $\sigma=1$), Galerkin method shows instabilities while all other methods are stable

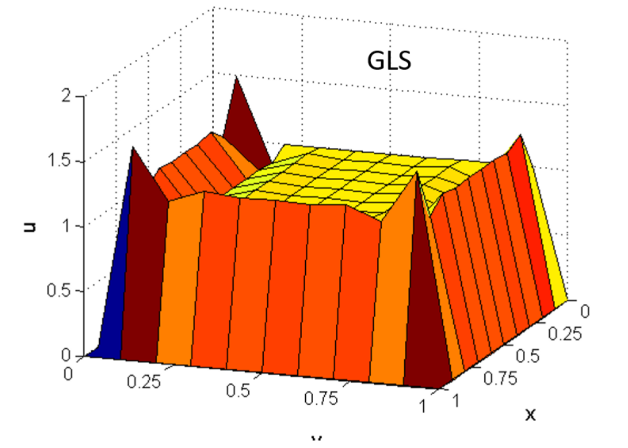


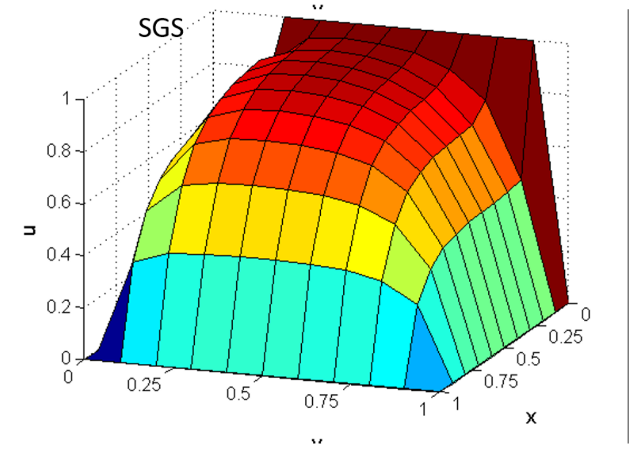
d. Reaction Dominated Case

Dirichlet Boundary Conditions (||a||=10^{-3}, ν=10^{-4}, σ=1), all methods show stabilize results.









Codes

1. Modifications to solve Convection-diffusion Reaction Equation
Linear Elements:

```
function [K,f] = FEM_system(X,T,referenceElement,example)
if method == 1
disp('Galerkin formulation');
tau = 0;
elseif method == 2
disp ('SUPG formulation');
Pe = a*h/(2*nu);
% alpha = coth(Pe)-1/Pe;
% tau_p = alpha*h/(2*a);
tau_p = h*(1 + 9/Pe^2 +(h*sigma/(2*a))^2)^(-1/2)/(2*a);
disp(strcat('Recommended value for the stabilization parameter =
',num2str(tau_p)));
tau = input('Stabilization parameter to be used (press return for using the
recommended one)= ');
if isempty(tau)
tau = tau_p;
end
elseif method == 3
disp('GLS formulation');
Pe = a*h/(2*nu);
% alpha = coth(Pe)-1/Pe;
% tau_p = alpha*h/(2*a);
tau_p = h*(1 + 9/Pe^2 +(h*sigma/(2*a))^2)^(-1/2)/(2*a);
disp(strcat('Recommended value for the stabilization parameter =
',num2str(tau_p)));
tau = input('Stabilization parameter to be used (press return for using the
recommended one)= ');
if isempty(tau)
tau = tau_p;
end
elseif method==4
% Artificial Diffusion
Pe_x = ax*h/(2*nu);
Pe_y = ay*h/(2*nu);
alpha_x = coth(Pe_x)-1/Pe_x;
alpha_y = coth(Pe_y)-1/Pe_y;
tau = h*(ax*alpha_x + ay*alpha_y)/2;
nubar_p = tau*a*a;
disp(strcat('Recommended value for the artificial diffusion =
',num2str(nubar_p)));
nubar = cinput('Artificial diffusion to be used', nubar_p);
if isempty(nubar)
nubar = nubar_p;
end
nu = nu + nubar;
tau=0;
end
```

```
function [Ke,fe] = EleMat(Xe,nen,ngaus,wgp,N,Nxi,Neta,method,tau,velo,nu)
if method == 1 % Galerkin
convi = ax*Nx+ay*Ny;
Ke = Ke + (N_ig'*convi + nu*(Nx'*Nx+Ny'*Ny) + sigma*N_ig'*N_ig)*dvolu;
aux = Isopar(Xe,N_ig);
f_ig = SourceTerm(aux);
fe = fe + N_ig'*(f_ig*dvolu);
elseif method == 2 % SUPG
convi = ax*Nx+ay*Ny;
Ke = Ke + (N_ig'*convi + nu*(Nx'*Nx+Ny'*Ny) + sigma*N_ig'*N_ig ...
+ tau*(convi'*convi + convi'*sigma*N_ig))*dvolu ;
aux = Isopar(Xe,N_ig);
f_ig = SourceTerm(aux);
fe = fe + (N_ig + tau*(ax*Nx+ay*Ny))'*(f_ig*dvolu);
elseif method == 3 % GLS
convi = ax*Nx + ay*Ny;
Ke = Ke + (N_ig'*convi + nu*(Nx'*Nx+Ny'*Ny) + sigma*N_ig'*N_ig ...
+ tau*((convi + sigma*N_ig)'*(convi + sigma*N_ig)))*dvolu ;%one term to be
added
aux = Isopar(Xe,N_ig);
f_ig = SourceTerm(aux);
fe = fe + (N_ig+ tau*(convi+sigma*N_ig))'*(f_ig*dvolu);
elseif method==4 %Artificial Diffusion
% Artifficial diffusion
Ke = Ke + (nu*(Nx'*Nx+Ny'*Ny) +
N_ig'*(ax*Nx+ay*Ny))*dvolu+sigma*(N_ig'*N_ig)*dvolu;
aux = N_ig*Xe;
f_ig = SourceTerm(aux);
fe = fe + N_ig'*(f_ig*dvolu);
```

2. Boundary Conditions in main.m

```
% BOUNDARY CONDITIONS
% Essential boundary conditions are imposed on the whole boundary
% using Lagrange multipliers
nodes_y0 = [1:nx+1]'; % Nodes on the boundary y=0
nodes_x1 = [2*(nx+1):nx+1:(ny+1)*(nx+1)]' ; % Nodes on the boundary x=1
nodes_y1 = [ny*(nx+1)+nx:-1:ny*(nx+1)+1]' ; % Nodes on the boundary y=1
nodes_x0 = [(ny-1)*(nx+1)+1:-(nx+1):nx+2]'; % Nodes on the boundary x=0
plot_BC;
disp ('On G1 solution is set to 1');
disp ('On G2 solution is set to 0');
disp ('On G3 either natural or essential homogeneous boundary conditions can
be imposed.')
BC = input ('Condiciones en G3 (1:Neumann, 2: Dirichlet): ');
if BC == 1
% nodes on which solution is u=1
nodesDir1 = [nodes_x0(1:ny-2)];
% nodes on which solution is u=0
nodesDir0 = [nodes_x0(ny-1);nodes_y0];
% Boundary condition matrix
C = [nodesDir1, ones(length(nodesDir1),1);
nodesDir0, zeros(length(nodesDir0),1)];
elseif BC == 2
% nodes on which solution is u=1
```

```
nodesDir1 = [nodes_x0(1:ny-2)];
% nodes on which solution is u=0
nodesDir0 = [nodes_x0(ny-1); nodes_y0; nodes_x1; nodes_y1 ];
% Boundary condition matrix
C = [nodesDir1, ones(length(nodesDir1),1);
nodesDir0, zeros(length(nodesDir0),1)];
else
error ('Error imposing boundary conditions' );
end
```

Quadratic Elements (GLS)

```
% GLS
Pe = a*h/(2*nu);
tau = h*(1 + 9/Pe^2 +(sigma*h)/(2*a))^(-1/2)/(2*a);
Ke = Ke + (nu*(Nx'*Nx+Ny'*Ny) +N_ig'*(ax*Nx+ay*Ny)+ N_ig'*sigma*N_ig + ...
tau*(ax*Nx+ay*Ny+sigma*N_ig)'*(ax*Nx+ay*Ny + N_ig*sigma))*dvolu;
for i=1:nedofV
if mod(i,2)==1
DKe(:,:,i) = DKe(:,:,i) + (N_ig'*(N_ig((i+1)/2)*Nx) ...
+ tau*(ax*Nx+ay*Ny)'*(N_ig((i+1)/2)*Nx) ...
+ tau*(N_ig((i+1)/2)*Nx)'*(ax*Nx+ay*Ny) )*dvolu ;
else
DKe(:,:,i) = DKe(:,:,i) + (N_ig'*(N_ig(i/2)*Ny) ...
+ tau*(ax*Nx+ay*Ny)'*(N_ig(i/2)*Ny) ...
+ tau*(N_ig(i/2)*Ny)'*(ax*Nx+ay*Ny) )*dvolu ;
end
end
f_ig = SourceTerm(a);
fe = fe + (N_ig+tau*(ax*Nx+ay*Ny+sigma*N_ig))'*(f_ig*dvolu);
for i=1:nedofV
if mod(i,2)==1
Dfe(:,i) = Dfe(:,i) +
(N_ig+tau*(ax*Nx+ay*Ny+sigma*N_ig))'*(Df_ig*dvolu)*N_ig((i+1)/2)^2*ue((i+1)/2
)/a;
else
Dfe(:,i) = Dfe(:,i) +
(N_ig+tau*(ax*Nx+ay*Ny+sigma*N_ig))'*(Df_ig*dvolu)*N_ig(i/2)^2*ve(i/2)/a;
end
end
```