close all

clear all

% Number of points

numpoint = 10;

% Velocity

Pe = 7;

% Diffusion coefficient

difcoef = 1;

% Domain length

lengthofdom = 1;

% Number of elements

numofelem = numpoint - 1;

% Element length

elength = lengthofdom/numofelem;

% Calculation of velocity

Velocity = (2\*Pe\*difcoef)/elength;

% Time array

timearray = 0:elength:lengthofdom;

% Analytical solution

phi\_exact = zeros(length(timearray), 1);

for i = 1:length(timearray)

phi\_exact(i) = (exp(Velocity\*timearray(i)/difcoef) - exp(Velocity\*lengthofdom/difcoef))/(1 - exp(Velocity\*lengthofdom/difcoef));

end

% Plot analytical solution

plot(timearray, phi\_exact, '-k', 'LineWidth', 1.2);

hold on

grid all

% Alpha values for different weighting schemes

alpha0 = 0; % standard Galerkin method

alpha1 = 1; % Petrov Galerkin method

alpha2 = 1 - (1/abs(Pe)); % alpha equals to alpha critical

alpha3 = alpha2\*0.5; % smaller than alpha critical

alpha4 = alpha2\*3; % greater than alpha critical

alpha5 = coth(abs(Pe)) - (1/abs(Pe)); % equals alpha optimal

% Array of alpha values

alpha = [alpha0 alpha1 alpha2 alpha3 alpha4 alpha5];

alpha = [alpha1 alpha2 alpha3 alpha4 alpha5];

% Node connectivity

lp = zeros(numofelem, 2);

for i = 1:numofelem

lp(i, 1) = i;

lp(i, 2) = i + 1;

end

% Initialize matrix

Convmat = zeros(numpoint, numpoint); % Convection matrix

Diffmat = zeros(numpoint, numpoint); % Diffusion matrix

for i = 1:numofelem

% Current element convection matrix

currentelem = Velocity \* 0.5 \* [-1 1; -1 1];

% Current element diffusion matrix

currentelemdiff = difcoef/elength \* [1 -1; -1 1];

% Current element extra diffusion matrix

currentextradiff = Velocity \* alpha(i)/2 \* [-1 1; -1 1]; % Extra diffusion matrix for Petrov Galerkin methods

% Global nodes of element

gp = lp(i, :);

% Global matrix

Convmat(gp, gp) = Convmat(gp, gp) + currentelem;

Diffmat(gp, gp) = Diffmat(gp, gp) + currentelemdiff;

extraDiff(gp, gp) = extraDiff(gp, gp) + currentextradiff;

end

% Boundary condition

phi = zeros(numpoint, 1);

phi(1) = 1; % Boundary condition at entry phi = 1 at inlet

phi(end) = 0; % Boundary condition at exit phi = 0 at outlet

% Matrix created

A = Convmat + Diffmat + extraDiff;

A\_red = A(2:numpoint-1, 2:numpoint-1);

% Reduced system

Rhs\_red = phi(2:numpoint-1);

Rhs\_red(1) = Rhs\_red(1) - A(2, 1);

newphi = A\_red \ Rhs\_red;

phi(2:numpoint-1) = newphi;

% Plotting

plot(timearray, phi, 'Marker', '\*', 'LineWidth', 0.8, 'MarkerSize', 10);

legend('Analytical Solution', 'Petrov Galerkin a = 1', 'a = acritical', 'a < acritical', 'a > acritical', 'a optimal', 'Location', 'north west');

title('Convection Diffusion for Peclet number(Pe)');

xlabel('Domain Length ');

ylabel('Scalar Quantity (\phi)');

% MATLAB Code for 1D Convection-Diffusion using Petrov-Galerkin Method

clear all;

close all;

% Parameters

numpoint = 10; % Number of points

Pe = 7; % Peclet number

u = 1; % Velocity

difcoef = 1; % Diffusion coefficient

L = 1; % Domain length

alpha = -1; % Upwinding factor (negative for upwinding)

% Mesh generation

x = linspace(0, L, numpoint);

h = x(2) - x(1);

% Shape functions and their derivatives

N = @(xi) [(1 - xi) / 2, (1 + xi) / 2];

dNdxi = [-1/2, 1/2];

% Assemble global stiffness matrix and load vector

K\_global = zeros(numpoint, numpoint);

F\_global = zeros(numpoint, 1);

% Gauss point for integration

gauss\_point = 0;

weight = 2;

% Assembly process

for element = 1:numpoint-1

local\_K = zeros(2,2);

local\_M = zeros(2,2);

% Integration using Gauss quadrature

xi = gauss\_point;

N\_val = N(xi);

dN\_dxi\_val = dNdxi;

% Jacobian of the transformation from reference element to physical element

J = h/2;

dN\_dx = dN\_dxi\_val / J;

% Compute element stiffness matrix

local\_K = local\_K + (u \* N\_val' \* dN\_dx + difcoef \* dN\_dx' \* dN\_dx) \* weight \* J;

% Petrov-Galerkin weighting

dN\_dx\_mod = dN\_dx + alpha \* abs(u) / (2 \* difcoef) \* dN\_dx;

local\_M = local\_M + (u \* N\_val' \* dN\_dx\_mod) \* weight \* J;

% Add local contribution to global matrix

K\_global(element:element+1, element:element+1) = ...

K\_global(element:element+1, element:element+1) + local\_K + local\_M;

end

% Apply boundary conditions

K\_global(1, :) = 0; K\_global(1, 1) = 1; F\_global(1) = 1; % Dirichlet BC at x=0

K\_global(end, :) = 0; K\_global(end, end) = 1; F\_global(end) = 0; % Dirichlet BC at x=L

% Solve the linear system

phi = K\_global \ F\_global;

% Post-processing: plot the results

plot(x, phi, 'o-', 'LineWidth', 2);

title('Solution of Convection-Diffusion Equation using Petrov-Galerkin FEM');

xlabel('Domain (x)');

ylabel('Scalar Variable (\phi)');

grid on;