Galerkin's method is a numerical method for solving differential equations using a weighted residual approach. It involves choosing a finite set of basis functions and finding the coefficients that minimize the residual error.

To solve the differential equation using Galerkin's method, we can follow these steps:

1. Choose a finite set of basis functions, in this case, we are given the following polynomial basis:

p1=x\*(1-x)

p2=x\*(1/2-x)\*(1-x)

p3=x\*(1/3-x)(2/3-x)(1-x)

p4=x\*(1/4-x)(1/2-x)(3/4-x)\*(1-x)

1. Assume a trial solution for u(x) as a linear combination of these basis functions multiplied by unknown coefficients. The trial solution is given by:

u(x) = a1p1 + a2p2 + a3p3 + a4p4

where a1, a2, a3, a4 are unknown coefficients to be determined.

1. Substitute the trial solution into the differential equation, and multiply both sides by each basis function pi, then integrate over the interval [0,1].

∫[0,1] pi\*d/dx( (1+x)du/dx) dx = - ∫[0,1] pix^2 dx

We can use integration by parts on the left-hand side to obtain:

[ pi\*(1+x)du/dx ]\_0^1 - ∫[0,1] d(pi(1+x)/dx \* du/dx) dx = - ∫[0,1] pi\*x^2 dx

Now, apply the boundary conditions u(0)=u(1)=0 to simplify the expression.

∫[0,1] d(pi\*(1+x)/dx \* du/dx) dx = ∫[0,1] pi\*x^2 dx

1. Using the properties of the basis functions and the boundary conditions, we can obtain a system of linear equations for the unknown coefficients a1, a2, a3, a4. The system of equations can be written in matrix form as:

[ K ] [ a ] = [ f ]

where K is the stiffness matrix, a is the vector of unknown coefficients, and f is the forcing vector.

The entries of the stiffness matrix and the forcing vector can be computed by evaluating the integrals from step 3.

1. Solve the system of linear equations to find the values of the unknown coefficients a1, a2, a3, a4.
2. Substitute the values of the coefficients into the trial solution to obtain the approximate solution for the differential equation.

The MATLAB code for solving the differential equation using Galerkin's method is as follows:

% Define the basis functions

p1 = @(x) x.\*(1-x);

p2 = @(x) x.\*(1/2-x).\*(1-x);

p3 = @(x) x.\*(1/3-x).\*(2/3-x).\*(1-x);

p4 = @(x) x.\*(1/4-x).\*(1/2-x).\*(3/4-x).\*(1-x);

% Define the integrand functions

f = @(x) -x.^2;

K = @(i,j) integral(@(x) (1+x).\*diff(p1(x)).\*diff(p2(x)), 0, 1);

F = @(i) integral(@(x) f(x).\*p(i,x), 0, 1);

% Set up the stiffness matrix and forcing vector

n = 4; % number of basis functions

Kmat = zeros(n,n); % stiffness matrix

Fvec = zeros(n,1); % forcing vector

% Compute the entries of the stiffness matrix and forcing vector

for i = 1:n

for j = 1:n

Kmat(i,j) = K(i,j);

end

Fvec(i) = F(i);

end

% Apply the boundary conditions

Kmat(1,:) = 0; Kmat(1,1) = 1; Fvec(1) = 0;

Kmat(n,:) = 0; Kmat(n,n) = 1; Fvec(n) = 0;

% Solve the system of linear equations

a = Kmat\Fvec;

% Compute the approximate solution

u = @(x) a(1)\*p1(x) + a(2)\*p2(x) + a(3)\*p3(x) + a(4)\*p4(x);

% Plot the approximate solution

x = linspace(0,1,100);

plot(x,u(x));

xlabel('x');

ylabel('u(x)');

title('Approximate solution using Galerkin''s method');

To compute the error between the approximate solution and the exact solution, we first need to find the exact solution to the differential equation. This can be done by integrating the differential equation twice and applying the boundary conditions. The exact solution is:

u(x) = (x/6) \* (x^2 - 3x + 2)

We can then compute the error between the approximate and exact solutions using the following code:

% Define the exact solution

u\_exact = @(x) (x/6).\*(x.^2 - 3\*x + 2);

% Compute the error between the approximate and exact solutions

x = linspace(0,1,100);

error = u\_exact(x) - u(x);

% Plot the approximate and exact solutions, and the error

plot(x,u(x),'b-',x,u\_exact(x),'r-',x,error,'g--');

legend('Approximate solution','Exact solution','Error');

xlabel('x');

ylabel('u(x)');

title('Approximate and exact solutions, and error');

% Compute the residual and plot it

residual = (1+x).\*diff(u(x),2) + x.^2;

figure;

plot(x,residual);

xlabel('x');

ylabel('Residual');

title('Residual plot');

This code will plot the approximate solution, exact solution, and the error between them. It will also compute and plot the residual. The residual is the left-hand side of the differential equation minus the right-hand side, evaluated at the approximate solution. Ideally, the residual should be close to zero everywhere.

Final Code :

% Define the basis functions

p1 = @(x) x.\*(1-x);

p2 = @(x) x.\*(1/2-x).\*(1-x);

p3 = @(x) x.\*(1/3-x).\*(2/3-x).\*(1-x);

p4 = @(x) x.\*(1/4-x).\*(1/2-x).\*(3/4-x).\*(1-x);

% Define the exact solution

u\_exact = @(x) (x/6).\*(x.^2 - 3\*x + 2);

% Define the integrand functions

f = @(x) -x.^2;

% Define the range of basis functions to use

n\_vec = [1, 2, 3, 4];

% Set up the figure for plotting the results

figure;

subplot(2,2,1);

% Loop over different numbers of basis functions

for i = 1:length(n\_vec)

n = n\_vec(i);

% Set up the stiffness matrix and forcing vector

Kmat = zeros(n,n); % stiffness matrix

Fvec = zeros(n,1); % forcing vector

% Compute the entries of the stiffness matrix and forcing vector

for j = 1:n

for k = 1:n

Kmat(j,k) = integral(@(x) (1+x).\*diff(eval(['p',num2str(j)]),2).\*eval(['p',num2str(k)]), 0, 1);

end

Fvec(j) = integral(@(x) f(x).\*eval(['p',num2str(j)])(x), 0, 1);

end

% Apply the boundary conditions

Kmat(1,:) = 0; Kmat(1,1) = 1; Fvec(1) = 0;

Kmat(n,:) = 0; Kmat(n,n) = 1; Fvec(n) = 0;

% Solve the system of linear equations

a = Kmat\Fvec;

% Compute the approximate solution

u = @(x) 0;

for j = 1:n

u = @(x) u(x) + a(j)\*eval(['p',num2str(j)])(x);

end

% Compute the error between the approximate and exact solutions

x = linspace(0,1,100);

error = u\_exact(x) - u(x);

% Compute the residual

residual = (1+x).\*diff(u(x),2) + x.^2;

% Plot the approximate and exact solutions, and the error

subplot(2,2,i);

plot(x,u(x),'b-',x,u\_exact(x),'r-',x,error,'g--');

legend('Approximate solution','Exact solution','Error');

xlabel('x');

ylabel('u(x)');

title(sprintf('%d basis functions', n));

% Plot the residual

figure;

plot(x,residual);

xlabel('x');

ylabel('Residual');

title(sprintf('Residual plot for %d basis functions', n));

end

This code will loop over different numbers of basis functions and compute the approximate solution, error, and residual for each. It will also plot the approximate and exact solutions, and the error, as well as the residual, for each number of basis functions. The results will be displayed in a 2x2 subplot figure.