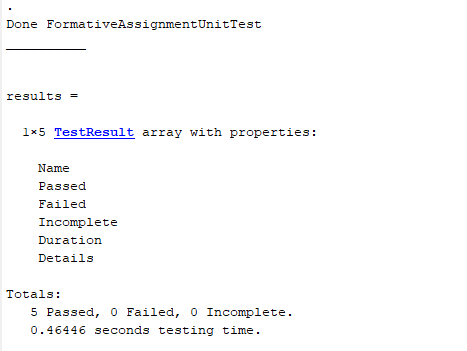
System modelling and simulation Coursework 1 – rrb32

1. Q1 – Local element diffusion matrix
   1. A)



Function to compute Diffusion LEM

%Function to calculate the diffusion local element matrix for a given

%mesh element

%Takes:

%D - Diffusion coefficient (int)

%eID - Element number (int)

%msh - Mesh data structure, generated using OneDimLinearMeshGen.m or

%Or OneDimSimpleRefinedMeshGen.m

function LocalElementmatrix = LaplaceElemMatrix(D,eID,msh)

%Extract J from msh structure

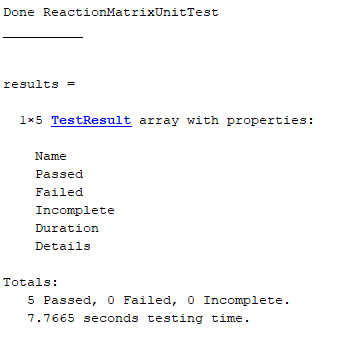
J = msh.elem(eID).J;

%Form local element matrix using equation derived in the notes

LocalElementmatrix = [ D/(2\*J) -D/(2\*J) ;

-D/(2\*J) D/(2\*J)];

* 1. B)



Function to compute Reaction LEM

%Function to calculate the reaction local element matrix for a given

%mesh element

%Takes:

%Lamda - Reaction coefficient (int)

%eID - Element number (int)

%msh - Mesh data structure, generated using OneDimLinearMeshGen.m or

%Or OneDimSimpleRefinedMeshGen.m

function LocalElementmatrix = ReactionMatrix(Lamda,eID,msh)

%Extract J from msh structure

J = msh.elem(eID).J;

%Form local element matrix using equation derived in the notes

LocalElementmatrix = [ (2\*Lamda\*J)/3 (Lamda\*J)/3 ;

(Lamda\*J)/3 (2\*Lamda\*J)/3 ];

Reaction LEM unit test

%Similarities between diffusion and reaction local element matrices allow

%lots of the provided code to be reused here:

%% Test 1: test symmetry of the matrix

% % Test that this matrix is symmetric. This is the same verification as for the

% diffusion LEM's

tol = 1e-14;

Lamda = 2; %Reaction coefficient

eID=1; %element ID

msh = OneDimLinearMeshGen(0,1,10);

elemat = ReactionMatrix(Lamda,eID,msh); %THIS IS THE FUNCTION YOU MUST WRITE

assert(abs(elemat(1,2) - elemat(2,1)) <= tol)

%% Test 2: test 2 different elements of the same size produce same matrix

% % Test that for two elements of an equispaced mesh, as described in the

% % lectures, the element matrices calculated are the same. This is the

% same verification as for the diffusion LEM's

tol = 1e-14;

Lamda = 5; %Reaction coefficient

eID=1; %element ID

msh = OneDimLinearMeshGen(0,1,10);

elemat1 = ReactionMatrix(Lamda,eID,msh);%THIS IS THE FUNCTION YOU MUST WRITE

eID=2; %element ID

elemat2 = ReactionMatrix(Lamda,eID,msh);%THIS IS THE FUNCTION YOU MUST WRITE

diff = elemat1 - elemat2;

diffnorm = sum(sum(diff.\*diff));

assert(abs(diffnorm) <= tol)

%% Test 3: test that one matrix is evaluted correctly

% % Test that element 1 of the three element mesh problem described in the lectures

% % the element matrix is evaluated correctly. This uses the example shown

% % from tutorial sheet 3.

tol = 1e-14;

Lamda = 9; %Reaction coefficient

eID=1; %element ID

msh = OneDimLinearMeshGen(0,1,3);

elemat1 = ReactionMatrix(Lamda,eID,msh);%THIS IS THE FUNCTION YOU MUST WRITE

elemat2 = [ 1 0.5; 0.5 1];

diff = elemat1 - elemat2; %calculate the difference between the two matrices

diffnorm = sum(sum(diff.\*diff)); %calculates the total squared error between the matrices

assert(abs(diffnorm) <= tol)

%% Test 4: test that different sized elements in a mesh are evaluted correctly - element 1

% % Test that elements in a non-equally spaced mesh are evaluated correctly

tol = 1e-14;

Lamda = 6; %Reaction coefficient

eID=1; %element ID

msh = OneDimSimpleRefinedMeshGen(0,1,5);

elemat1 = ReactionMatrix(Lamda,eID,msh);%THIS IS THE FUNCTION YOU MUST WRITE

elemat2 = [ 1 0.5; 0.5 1];

diff = elemat1 - elemat2; %calculate the difference between the two matrices

diffnorm = sum(sum(diff.\*diff)); %calculates the total squared error between the matrices

assert(abs(diffnorm) <= tol)

%% Test 5: test that different sized elements in a mesh are evaluted correctly - element 4

% % Test that elements in a non-equally spaced mesh are evaluated correctly

tol = 1e-14;

Lamda = 48; %Reaction coefficient

eID=4; %element ID

msh = OneDimSimpleRefinedMeshGen(0,1,5);

elemat1 = ReactionMatrix(Lamda,eID,msh);%THIS IS THE FUNCTION YOU MUST WRITE

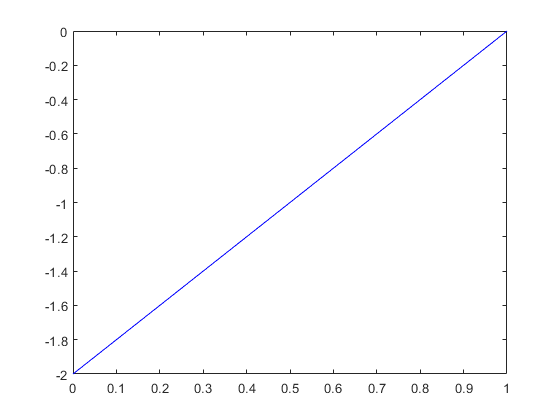
elemat2 = [ 1 0.5; 0.5 1];

diff = elemat1 - elemat2; %calculate the difference between the two matrices

diffnorm = sum(sum(diff.\*diff)); %calculates the total squared error between the matrices

assert(abs(diffnorm) <= tol)

1. Q2) Solving Laplaces eq.
   1. A) Plot of 2(x-1)



Perfect accuracy as this is a linear equation. Linear interpolation attracts 0 error as a result.

B) Laplace solver code

Called with: SolveLaplace(1,0,5,'VN',2,'DL',0)

%Function to solve Laplace's equation for given parameters of diffusion and

%reaction coefficients. As a Laplacian problem the source terms are 0.

%

%Takes the following arguments:

%

%D - Diffusion Co-efficient (Float)

%Lamda - Reaction Co-efficient (Float)

%NNodes - Number of nodes in global mesh (NElements = NNodes - 1) (Int)

%BC0 - Type of node 0 boundary condition, 'DL' for Dirichlet or 'VN' for Von

%Nuemman (Str)

%BC0Val - Value of c or dc/dx for node 0 boundary condition (Float)

%BC1 - Node 1 boundary condition, same format as BC0

%BC1Val - Value of c or dx/dx for node 1 boundary condition (Float)

%

%The solution is plotted against a known analytical solution for:

%SolveLaplace(1,-9,N,'DL',0,'DL',1) - Any positive integer N.

%E.g. SolveLaplace(1,-9,5,'DL',0,'DL',1)

%Note that the domain is currently hardcoded from x = 0 to x = 1

function [SolutionVector, Domain] = SolveLaplace(D,Lamda,NNodes,BC0,BC0Val,BC1,BC1Val)

%Set domain

xmin = 0;

xmax = 1;

% Initialise mesh

Mesh = OneDimLinearMeshGen(xmin,xmax,NNodes-1); % Elements will also be N-1 ;

%Size of global mesh effects local element values due to varying J scaling

GMatrix = zeros(NNodes,NNodes);

SourceVector = zeros(NNodes,1); % Source term is all 0s for laplacian eq.

%Form local element matrices as struct

for idx = 1: length(GMatrix) -1

% Generate diffusion local elements and populate global matrix

LocalMatrix = LaplaceElemMatrix(D,idx,Mesh);

GMatrix(idx:idx+1,idx:idx+1) = GMatrix(idx:idx+1,idx:idx+1) + LocalMatrix ;

% Generate reaction local elements and populate global matrix

LocalMatrix = ReactionMatrix(Lamda,idx,Mesh);

GMatrix(idx:idx+1,idx:idx+1) = GMatrix(idx:idx+1,idx:idx+1) - LocalMatrix ;

end

%Form source vector - Is there a more efficient method for this?

for idx = 1 : length(GMatrix)

if idx == 1

SourceVector(idx) = SourceVector(idx) \* Mesh.elem(idx).J;

elseif idx == length(GMatrix - 1)

SourceVector(idx) = SourceVector(idx) \* Mesh.elem(idx-1).J;

else

SourceVector(idx) = SourceVector(idx) \* (Mesh.elem(idx-1).J + Mesh.elem(idx).J);

end

end

% I dont think that this structure will cause problems with overwriting

% with Dirichlet boundary conditions, can you please confirm this?

%Enforce boundary conditions at node 0

switch BC0

case 'VN'

% VN

% Specifically for dc/dx = 2 at x =0

% Goes to -2 in source vector i.e. this sets the gradient of the solution

SourceVector(1) = SourceVector(1) + -BC0Val; % Need to add for VN

case 'DL'

% Dirichlet

% Specifically for c = 0 at at x = 1

GMatrix(1,:) = 0;

GMatrix(1,1) = 1;

SourceVector(1) = BC0Val; % c is 0 in this case % Need for overwrite for DC

end

%Enforce boundary conditions at node 1

switch BC1

case 'VN'

% VN

% Specifically for dc/dx = 2 at x =0

% Goes to -2 in source vector i.e. this sets the gradient of the solution

SourceVector(end) = SourceVector(end) + -BC1Val; % Need to add for VN

case 'DL'

% Dirichlet

% Specifically for c = 0 at at x = 1

GMatrix(end,:) = 0;

GMatrix(end) = 1;

SourceVector(end) = BC1Val; % c is 0 in this case % Need for overwrite for DC

end

%Solve matrix using matlab inbuilt matrix division

SolutionVector = GMatrix\SourceVector;

Domain = linspace(xmin,xmax,length(SolutionVector));

%Plot FEM solution

plot(Domain, SolutionVector, 'b')

%Add parameters to plot

%Generate analytical solution to compare with FEM solution

E = exp(1);

P = linspace(0,1,100);

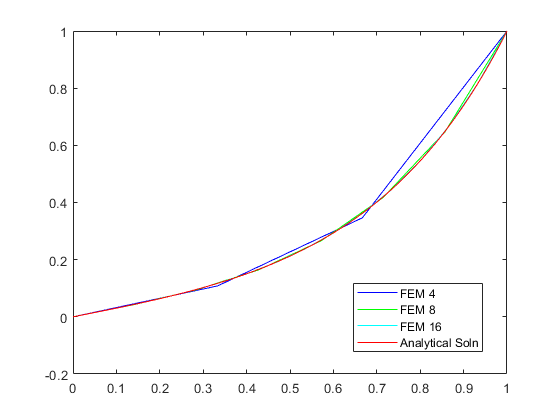
Ce = (E^3)/((E^6)-1) \* (E.^(3\*P) - E.^(-3\*P));

%Plot analytical solution to compare with FEM solution

%hold on

%plot(P,Ce,'r')

1. Q3 Laplace solver
   1. A) Plot of FEM vs analytical soln.



* 1. B) Source code for Laplace solver

There are a number of opportunities to implement sub-functions throughout this function, but I thought that for the sake of a few lines, it detracts from the readability to do this. Id appreciate some advice as to how desirable this is.

Plots generated by calling: SolveLaplace(1,-9,N,'DL',0,'DL',1) with N = [4,8,16]

%Function to solve Laplace's equation for given parameters of diffusion and

%reaction coefficients. As a Laplacian problem the source terms are 0.

%

%Takes the following arguments:

%

%D - Diffusion Co-efficient (Float)

%Lamda - Reaction Co-efficient (Float)

%NNodes - Number of nodes in global mesh (NElements = NNodes - 1) (Int)

%BC0 - Type of node 0 boundary condition, 'DL' for Dirichlet or 'VN' for Von

%Nuemman (Str)

%BC0Val - Value of c or dc/dx for node 0 boundary condition (Float)

%BC1 - Node 1 boundary condition, same format as BC0

%BC1Val - Value of c or dx/dx for node 1 boundary condition (Float)

%

%The solution is plotted against a known analytical solution for:

%SolveLaplace(1,-9,N,'DL',0,'DL',1) - Any positive integer N.

%E.g. SolveLaplace(1,-9,5,'DL',0,'DL',1)

%Note that the domain is currently hardcoded from x = 0 to x = 1

function [SolutionVector, Domain] = SolveLaplace(D,Lamda,NNodes,BC0,BC0Val,BC1,BC1Val)

%Set domain

xmin = 0;

xmax = 1;

% Initialise mesh

Mesh = OneDimLinearMeshGen(xmin,xmax,NNodes-1); % Elements will also be N-1 ;

%Size of global mesh effects local element values due to varying J scaling

GMatrix = zeros(NNodes,NNodes);

SourceVector = zeros(NNodes,1); % Source term is all 0s for laplacian eq.

%Form local element matrices as struct

for idx = 1: length(GMatrix) -1

% Generate diffusion local elements and populate global matrix

LocalMatrix = LaplaceElemMatrix(D,idx,Mesh);

GMatrix(idx:idx+1,idx:idx+1) = GMatrix(idx:idx+1,idx:idx+1) + LocalMatrix ;

% Generate reaction local elements and populate global matrix

LocalMatrix = ReactionMatrix(Lamda,idx,Mesh);

GMatrix(idx:idx+1,idx:idx+1) = GMatrix(idx:idx+1,idx:idx+1) - LocalMatrix ;

end

%Form source vector - Is there a more efficient method for this?

for idx = 1 : length(GMatrix)

if idx == 1

SourceVector(idx) = SourceVector(idx) \* Mesh.elem(idx).J;

elseif idx == length(GMatrix - 1)

SourceVector(idx) = SourceVector(idx) \* Mesh.elem(idx-1).J;

else

SourceVector(idx) = SourceVector(idx) \* (Mesh.elem(idx-1).J + Mesh.elem(idx).J);

end

end

% I dont think that this structure will cause problems with overwriting

% with Dirichlet boundary conditions, can you please confirm this?

%Enforce boundary conditions at node 0

switch BC0

case 'VN'

% VN

% Specifically for dc/dx = 2 at x =0

% Goes to -2 in source vector i.e. this sets the gradient of the solution

SourceVector(1) = SourceVector(1) + -BC0Val; % Need to add for VN

case 'DL'

% Dirichlet

% Specifically for c = 0 at at x = 1

GMatrix(1,:) = 0;

GMatrix(1,1) = 1;

SourceVector(1) = BC0Val; % c is 0 in this case % Need for overwrite for DC

end

%Enforce boundary conditions at node 1

switch BC1

case 'VN'

% VN

% Specifically for dc/dx = 2 at x =0

% Goes to -2 in source vector i.e. this sets the gradient of the solution

SourceVector(end) = SourceVector(end) + -BC1Val; % Need to add for VN

case 'DL'

% Dirichlet

% Specifically for c = 0 at at x = 1

GMatrix(end,:) = 0;

GMatrix(end) = 1;

SourceVector(end) = BC1Val; % c is 0 in this case % Need for overwrite for DC

end

%Solve matrix using matlab inbuilt matrix division

SolutionVector = GMatrix\SourceVector;

Domain = linspace(xmin,xmax,length(SolutionVector));

%Plot FEM solution

plot(Domain, SolutionVector, 'c')

%Add parameters to plot

%Generate analytical solution to compare with FEM solution

E = exp(1);

P = linspace(0,1,100);

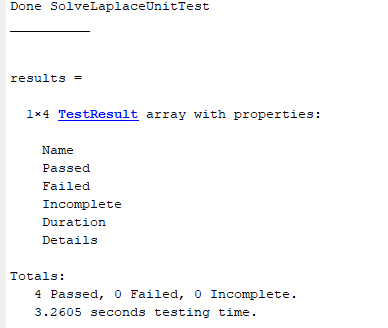
Ce = (E^3)/((E^6)-1) \* (E.^(3\*P) - E.^(-3\*P));

%Plot analytical solution to compare with FEM solution

hold on

plot(P,Ce,'r')

* 1. C) Unit test performance of laplace solver



* 1. D) Unit test for laplace solver

%% Test 1: test that 1st Dirichlet boundary condition is implemented correctly

% %

tol = 1e-14;

D = 1; %Diffusion Coefficient

Lamda = -9; %Reaction Coefficient

NNodes = 5;

BC0 = 'DL';

BC0Val = 0;

BC1 = 'DL';

BC1Val = 1;

[Solution, Domain] = SolveLaplace(D,Lamda,NNodes,BC0,BC0Val,BC1,BC1Val);

assert(abs(Solution(1) - BC0Val)< tol)

%% Test 2: test that 2nd Dirichlet boundary condition is implemented correctly

% %

tol = 1e-14;

D = 1; %Diffusion Coefficient

Lamda = -9; %Reaction Coefficient

NNodes = 5;

BC0 = 'DL';

BC0Val = 0;

BC1 = 'DL';

BC1Val = 1;

[Solution, Domain] = SolveLaplace(D,Lamda,NNodes,BC0,BC0Val,BC1,BC1Val);

assert(abs(Solution(end) - BC1Val)< tol)

%% Test 3: test that a Von Neumann BC is implemented correctly by verifying the gradient for a linear system

% %

tol = 1e-14;

D = 1; %Diffusion Coefficient

Lamda = 0; %Reaction Coefficient

NNodes = 5;

BC0 = 'VN';

BC0Val = 9.4;

BC1 = 'DL';

BC1Val = 0;

[Solution, Domain] = SolveLaplace(D,Lamda,NNodes,BC0,BC0Val,BC1,BC1Val);

grad = (Solution(end) - Solution(1))/(Domain(end) - Domain(1));

assert(abs(grad - BC0Val) < tol)

%% Test 4: test that with a very fine mesh that node points are "Close" to a known analytical solution

% % Is it neccesary to use an L2 Norm method to verify this more robustly?

%Have to reduce tolerance as we are dealing with truncation errors that are

%orders of magnitude larger than rounding errors

tol = 1e-3;

D = 1; %Diffusion Coefficient

Lamda = -9; %Reaction Coefficient

NNodes = 101; %Convenient for matching to 100 point domain

BC0 = 'DL';

BC0Val = 0;

BC1 = 'DL';

BC1Val = 1;

%FEM solution

[Solution, Domain] = SolveLaplace(D,Lamda,NNodes,BC0,BC0Val,BC1,BC1Val);

%Analytical solution

Exp = exp(1);

P = linspace(0,1,101);

Ce = (Exp^3)/((Exp^6)-1) \* (Exp.^(3\*P) - Exp.^(-3\*P));

Error = Ce - Solution';

TotalError = sum(Error);

assert(abs(TotalError) < tol)

close all % Close graphs that are usually plotted