

# Assignment 1: Static MATLAB-Based FEM Modelling

Note: I have all the code and figures and analysis that I need, I just haven't quite finished compiling the report. This is what I have so far. I'm compiling the rest now and I'll upload it as soon as I can.

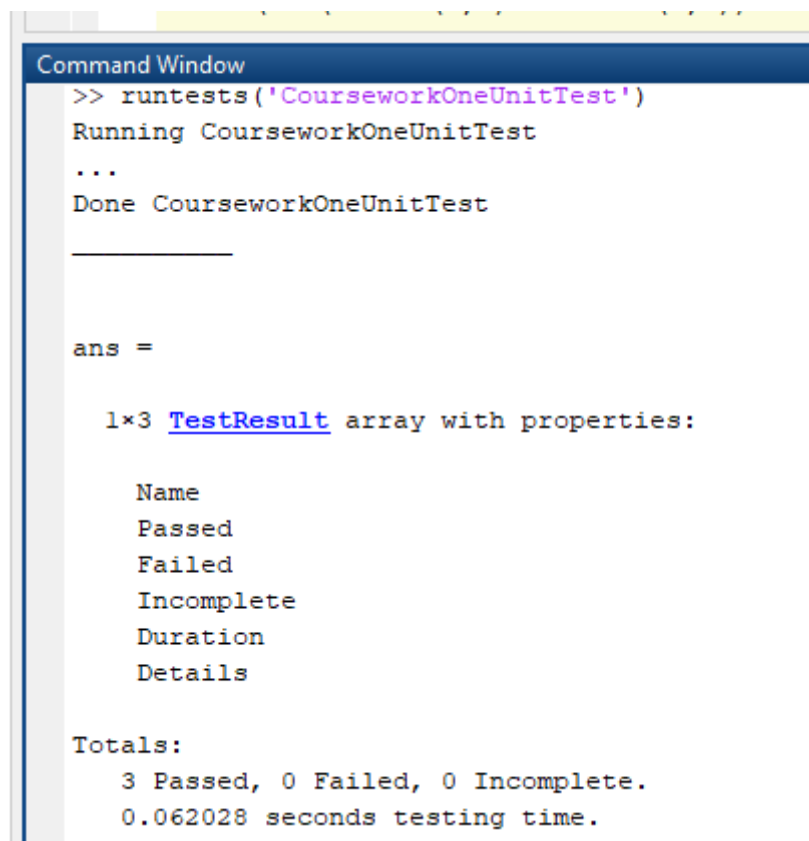
## Summary

During this assignment, an FEM-based simulation tool was developed to solve the static diffusion-reaction equation in a 1D mesh.

## Part 1: Software Verification and Analytical Testing

### a. Function to calculate a local element matrix for the Diffusion term

A function, LaplaceElemMatrix, was written to calculate a local element matrix for the Diffusion term. The source code can be found in the Appendix. The runtests() command was used to run CourseworkOneUnitTest and the results were as shown in Figure 1.



```

Command Window
>> runtests('CourseworkOneUnitTest')
Running CourseworkOneUnitTest
...
Done CourseworkOneUnitTest

ans =

1x3 TestResult array with properties:
    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

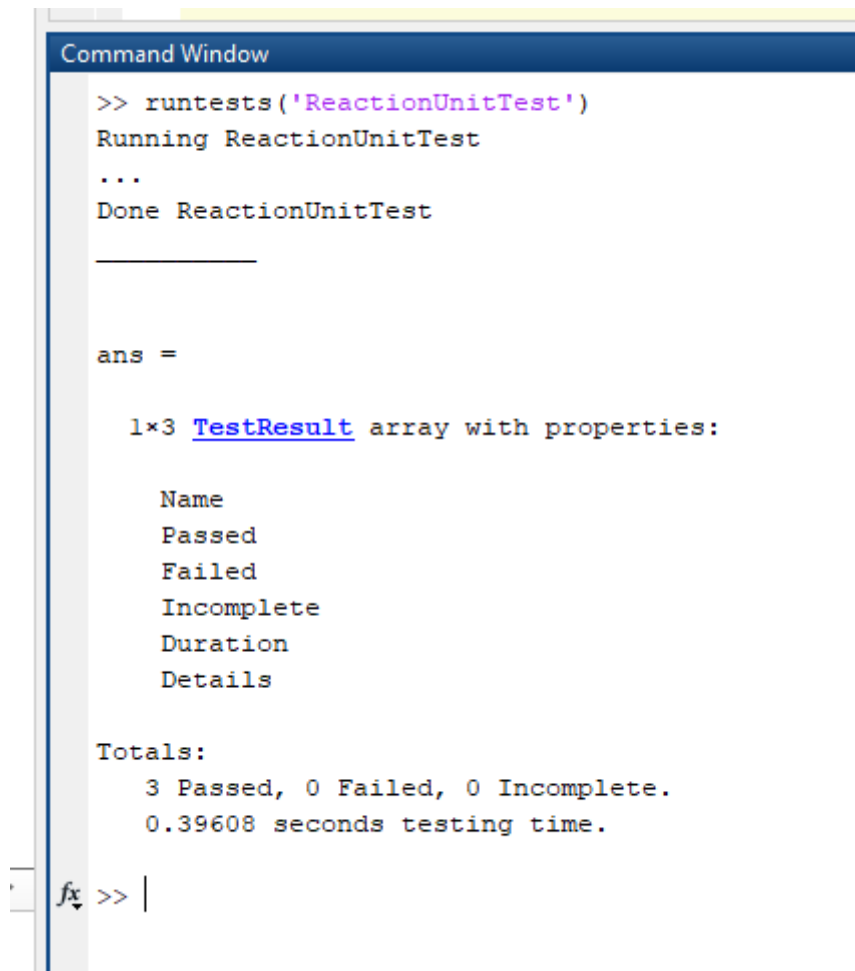
Totals:
    3 Passed, 0 Failed, 0 Incomplete.
    0.062028 seconds testing time.
  
```

Figure 1: Screenshot showing that the function passed the unit test.

### b. Function to calculate a local element matrix for the Reaction term

A function, ReactionElemMatrix, was written to calculate a local element matrix for the Reaction term. ReactionUnitTest was written to test that: the matrix was symmetrical, the same matrix was generated for the same size elements, and that one specific matrix was generated correctly, using

the example from the tutorials. The source code for both the function and the unit test can be found in the Appendix. The results of the unit test are as shown in Figure 2.



```

Command Window

>> runtests('ReactionUnitTest')
Running ReactionUnitTest
...
Done ReactionUnitTest

-----

ans =

    1x3 TestResult array with properties:

        Name
        Passed
        Failed
        Incomplete
        Duration
        Details

Totals:
    3 Passed, 0 Failed, 0 Incomplete.
    0.39608 seconds testing time.

fx >> |

```

Figure 2: Screenshot showing that the function passed the unit test.

### c. FEM Solver

An FEM solver was written based on the above functions to solve the static diffusion-reaction equation in a 1D mesh. The source code can be found in the Appendix.

#### Pseudo-Code

The pseudo-code for the FEM solver was as follows:

1. Initialise mesh
2. Define material coefficients
3. Create & initialise the global matrix and vector to zero
4. Loop over elements: calculate local element matrices and add to correct location in global matrix
5. Loop over elements: calculate local element vectors and add to correct location in global vector
6. Apply boundary conditions to global matrix and/or vector
7. Solve the final matrix system
8. Plot the solution vector

### Function to calculate a local element vector for the Source term

The element vector was initially calculated inline with the FEM solver function because it was relatively simple, but when it was later modified to allow for a linear source term, it became more complicated, and so a function was written to calculate it: ElemVector. The source code can be found in the Appendix.

### Analytical Test One – Laplace’s Equation

A script was written to solve Laplace’s Equation for the 1D 4 element mesh shown in Figure 3, using the FEMsolver function, with various boundary conditions. The source code can be found in the Appendix.

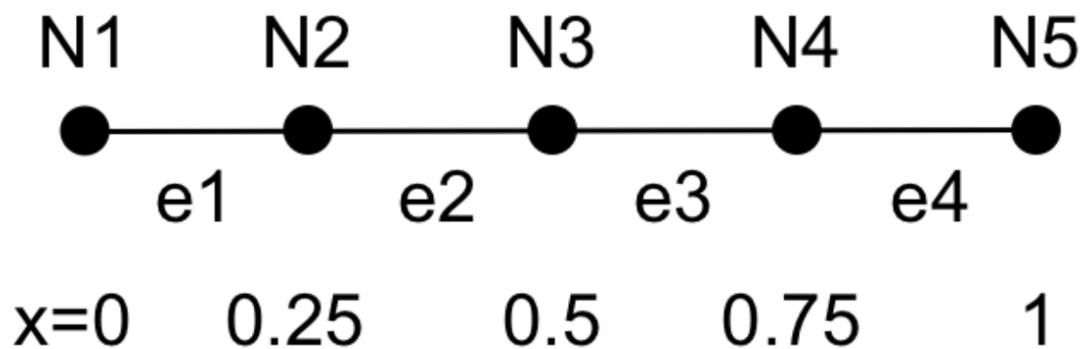


Figure 3: 1D 4 element mesh

#### i. Two Dirichlet Boundary Conditions

First, the equation was solved with two Dirichlet boundary conditions. Figure 4 shows the analytical solution compared to the numerical solution. The numerical solution is 100% accurate. This is because the analytical solution is linear, so there are no residual errors due to using linear basis functions.

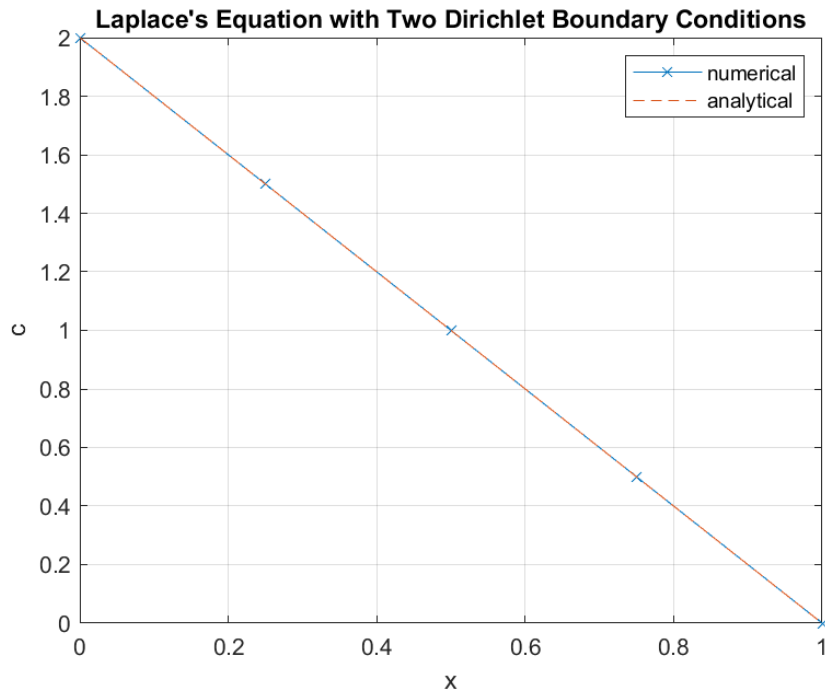


Figure 4: Analytical Test 1i

ii. *One Neumann and One Dirichlet Boundary Conditions*

Next, the equation was solved with one Dirichlet and one Neumann boundary condition. This had the effect of making the gradient of the whole solution equal to the gradient of the Neumann boundary condition at  $x=1$ . This is because the solution is linear so it can only have one gradient. Figure 5 shows the plot of the numerical solution.

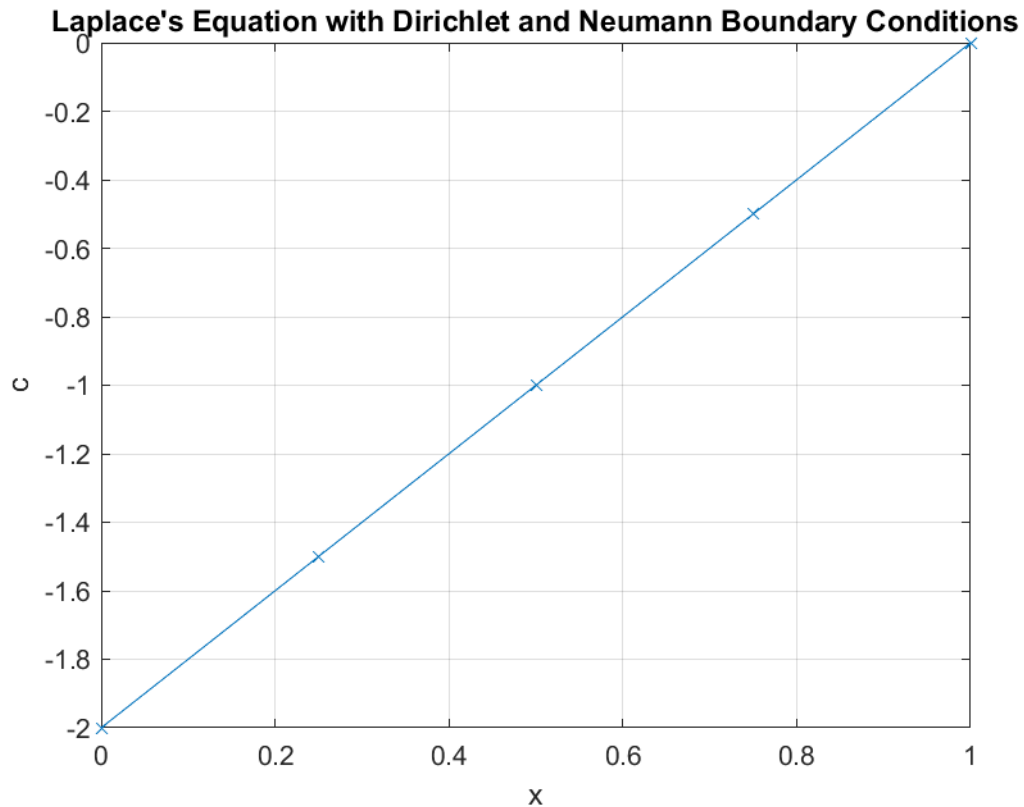


Figure 5: Test 1ii

### Analytical Test Two – Diffusion-Reaction Equation

Next, the solver was used to solve the Diffusion-Reaction Equation to check that the reaction term was coded correctly. The source code can be found in the same `analytical_tests` script in the Appendix as tests 1i and 1ii.

#### iii. Two Dirichlet Boundary Conditions

The equation was solved with two Dirichlet boundary conditions. Figure 6 shows that the numerical solution converges on the analytical solution as mesh resolution increases, and that it is indistinguishable by eye from the analytical solution at  $Ne=50$ .

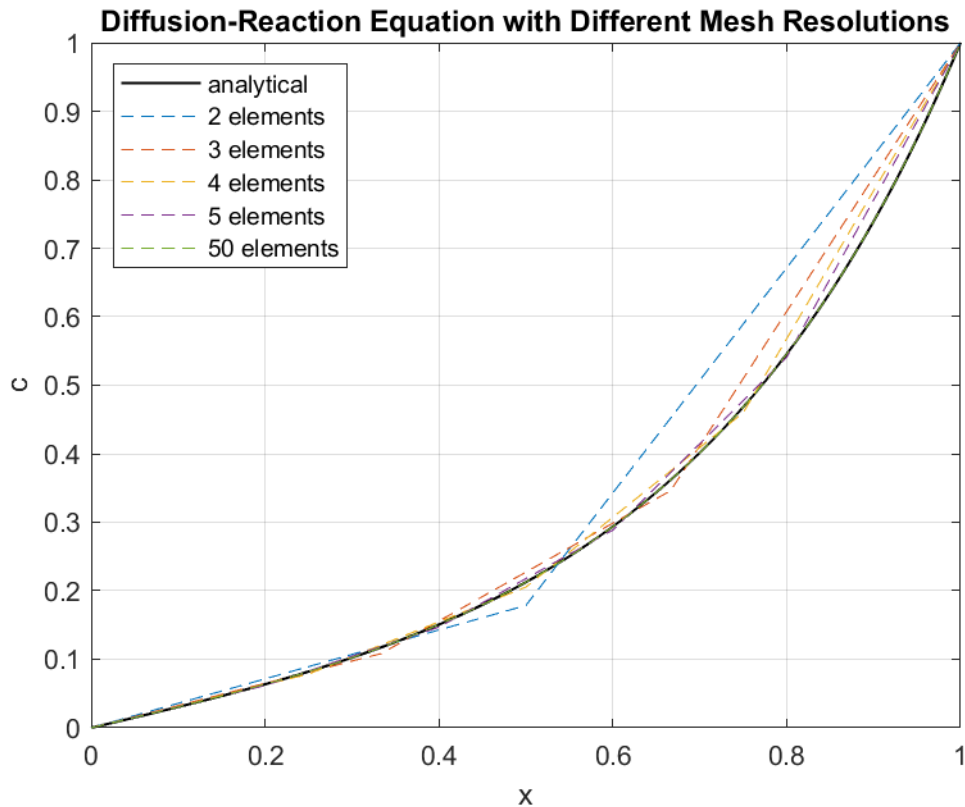


Figure 6: Test 1iii

## Part 2: Modelling and Simulation Results

Thermophoresis is a phenomenon in which particles of different types diffuse differently through a medium depending on the local temperature gradient that they are exposed to. The particles diffuse at a rate that is negatively proportional to the local temperature gradient i.e. the steeper the negative temperature gradient, the faster the particles will diffuse. In biomaterial processing this effect can be harnessed to create a spatially varying concentration of these particles, which causes a spatially varying stiffness to be created in the biomaterial. This is useful for growing cell cultures under biologically realistic conditions.

In Part 2 the code was used to model steady-state heat transfer through a material that is filled with small diameter heating channels. By varying the temperature of the heating liquid, the flow rate, or the spatial distribution of temperature in these channels, different temperature profiles can be obtained for the thermophoresis-based manufacturing process. The behaviour of this system was investigated and characterised for its different parameters.

Figure 7 shows the setup and the parameters to be investigated. The source code for the entire case study can be found in the Appendix under 'ThermophoresisCaseStudy'.

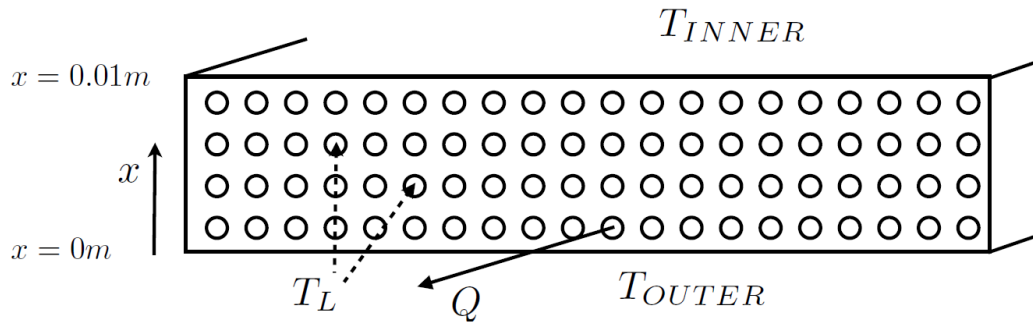


Figure 7: Thermophoresis Parameters

#### a. Constant Source Term

The initial model of the system was as follows:

$$k \frac{\partial^2 T}{\partial x^2} + Q(T_L - T) = 0$$

By comparing it to the equation that had been derived previously,  $k$  can be equated with the diffusion coefficient  $D$ ,  $Q$  can be equated with the reaction coefficient  $\lambda$ , and  $QT_L$  can be equated with the source term  $f$ .

#### i. Parameter Space Study

The code was used to model the system with various combinations of  $Q$  and  $T_L$ . All other parameters were held constant. The results are shown in Figures 8, 9, and 10.

The temperature gradient is always

It can be deduced from the equation above that the curvature of the temperature with respect to  $T$  is proportional to the difference between the temperature of the material and the temperature of the liquid,  $T - T_L$ .

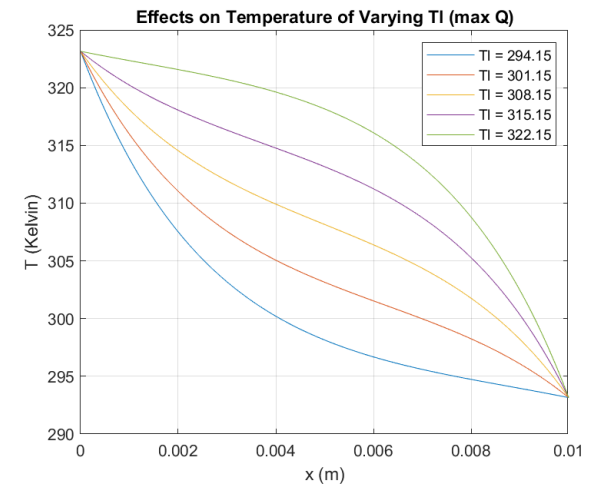
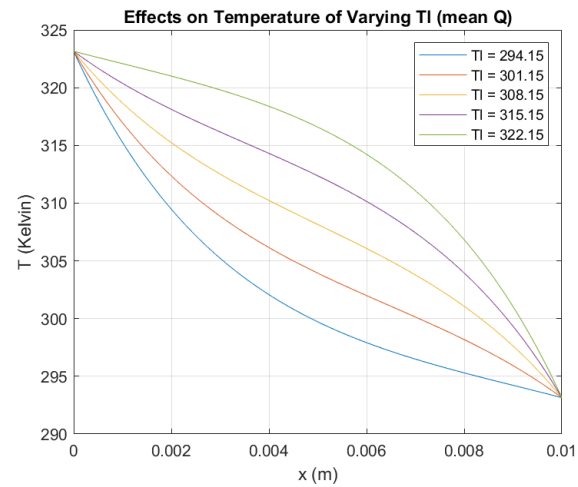
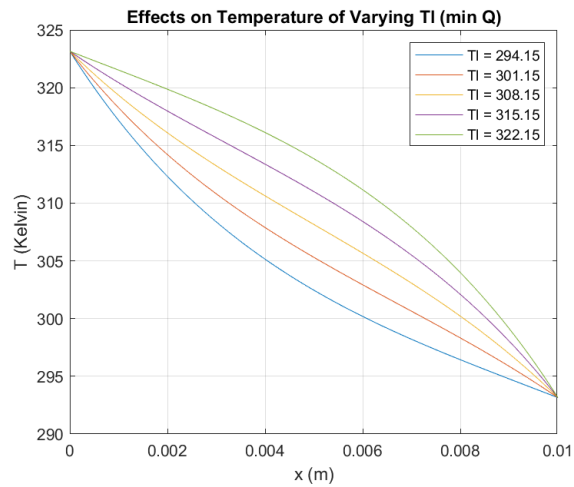
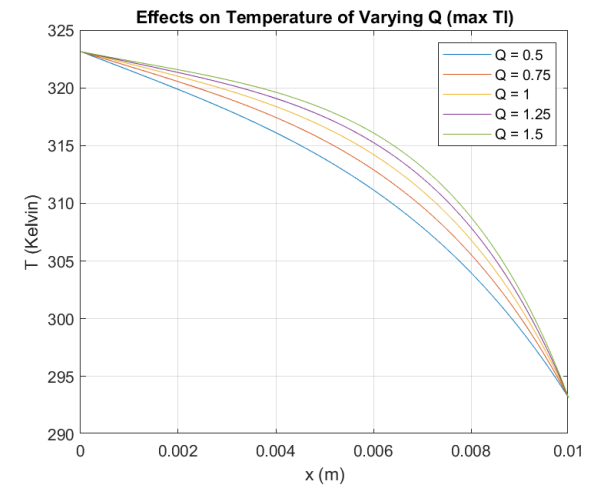
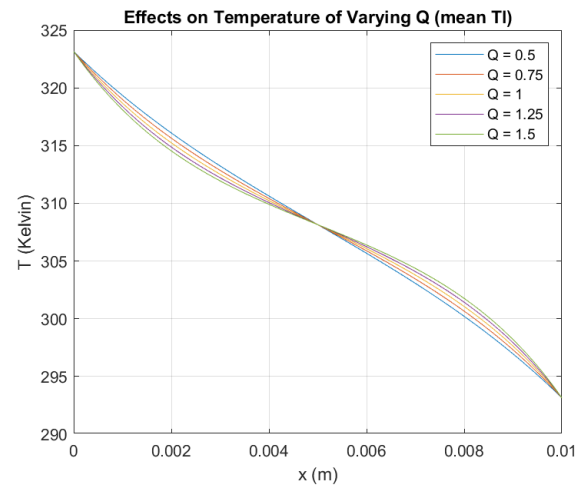
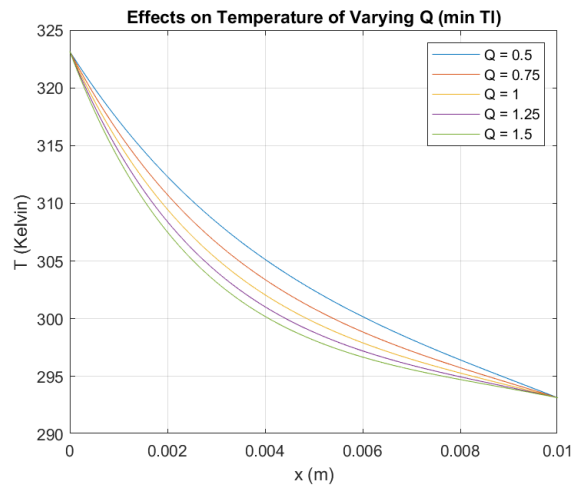


Figure 8ai (top left) to 8biii (bottom right): Effects on Temperature of Varying Q and  $T_L$



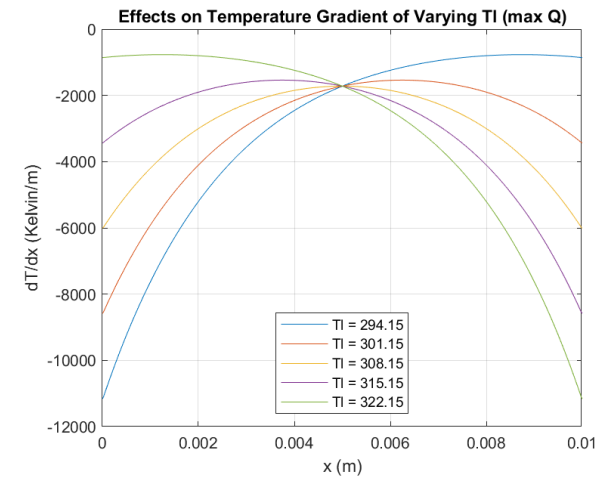
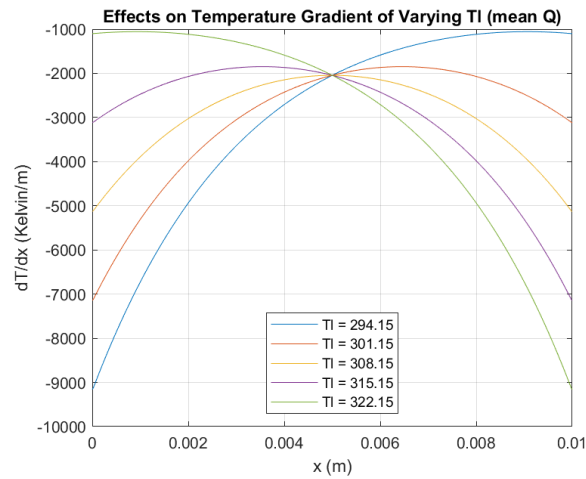
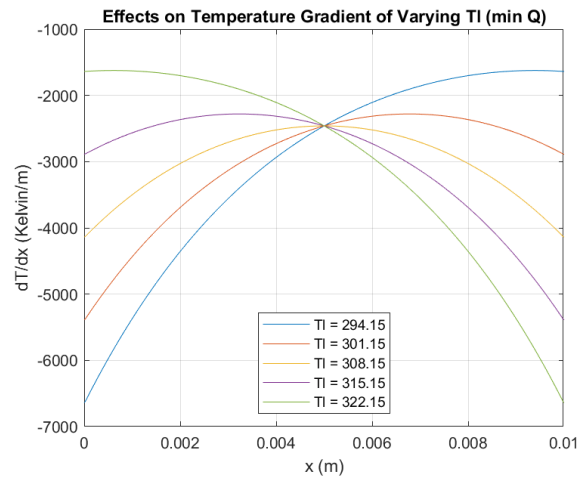
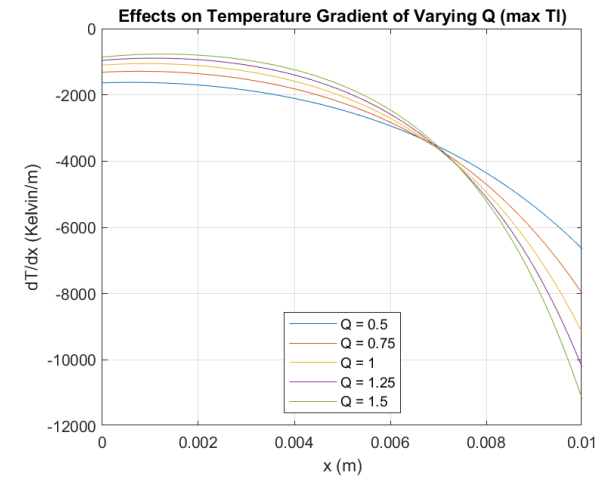
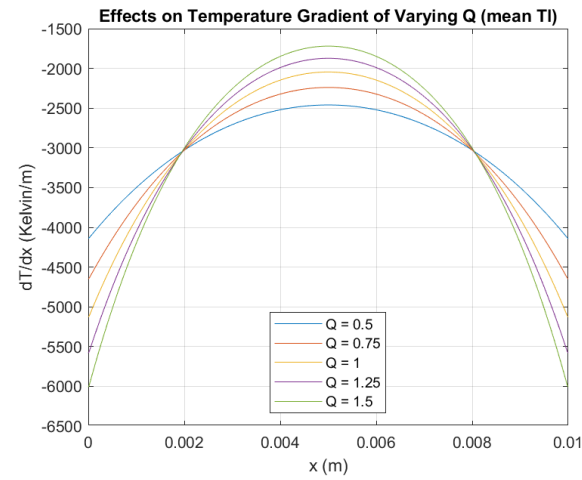
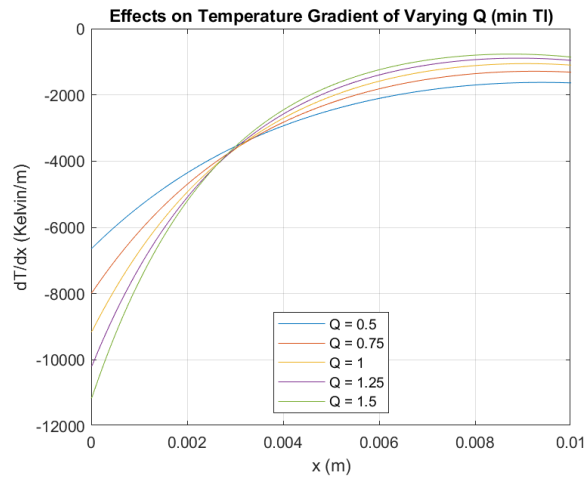


Figure 9ai (top left) to 9biii (bottom right): Effects on Temperature Gradient of Varying  $Q$  and  $T_L$

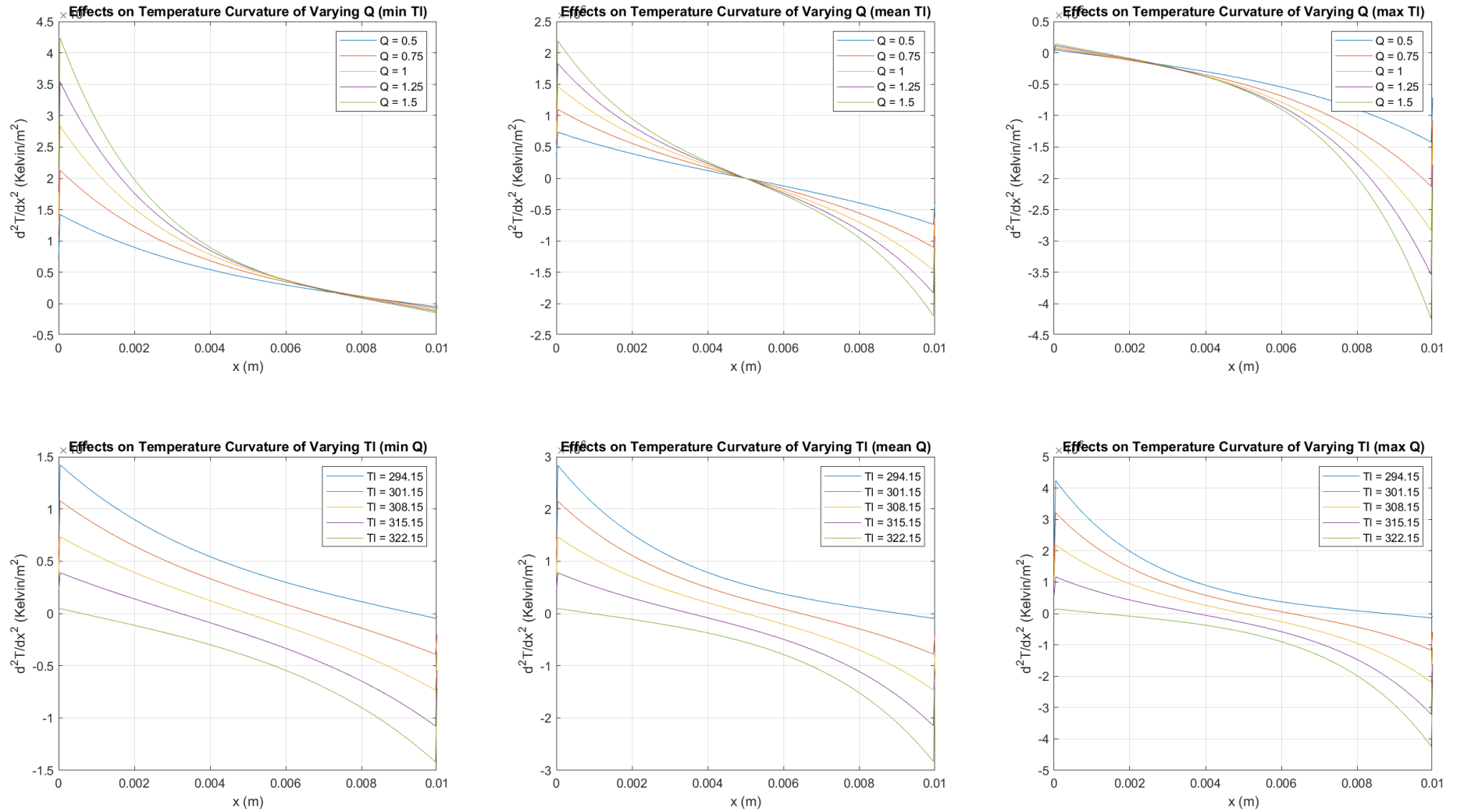


Figure 10 ai (top left) to 10biii (bottom right): Effects on Temperature Curvature of Varying  $Q$  and  $T_l$

ii. Predictions of Particle Diffusion due to Thermophoresis

Lorem ipsum

iii. Mesh Resolution Study

Lorem ipsum

b. Linear Source Term

Lorem ipsum

i. Deriving the analytical expression for the modified Source Vector

Lorem ipsum

ii. Implementation and Investigation

Lorem ipsum

## Conclusion

Lorem ipsum

## Appendix: Source Code

### LaplaceElemMatrix function

```
function L = LaplaceElemMatrix(D,eN,mesh)
% function to calculate local element matrix for diffusion term

x = mesh.elem(eN).x;
J = mesh.elem(eN).J;
dXiX = 2/(x(2)-x(1));
dPhiXi = [-1/2 1/2];

if isnumeric(D)
    L = zeros(2,2);
else
    L = sym(zeros(2,2)); % allows for symbolic D if desired
end

for n = 1:2
    for m = 1:2
        L(n,m) = 2*D* dPhiXi(n)*dXiX * dPhiXi(m)*dXiX * J;
    end
end

end
```

### ReactionElemMatrix function

```
function R = ReactionElemMatrix(lambda,eN,mesh)
% function to calculate local element matrix for diffusion term

J = mesh.elem(eN).J;
Int = [2/3 1/3; 1/3 2/3];

R = J * lambda * Int;

end
```

### ReactionUnitTest script

```
% Test 1: test symmetry of the matrix
% Test that this matrix is symmetric
tol = 1e-14;
lambda = 2; % reaction coefficient
eID=1; % element ID
msh = OneDimLinearMeshGen(0,1,10);

elemat = ReactionElemMatrix(lambda,eID,msh);

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
tol = 1e-14;
lambda = 5; % reaction coefficient
eID=1; % element ID
msh = OneDimLinearMeshGen(0,1,10);
```

```

elemat1 = ReactionElemMatrix(lambda,eID,msh);

eID=2; % element ID

elemat2 = ReactionElemMatrix(lambda,eID,msh);

diff = elemat1 - elemat2;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)

%% Test 3: test that one matrix is evaluated correctly
% % Test that element 1 of the three element mesh problem described in the
tutorials
% % the element matrix is evaluated correctly
tol = 1e-14;
lambda = 1; % reaction coefficient
eID=1; % reaction ID
msh = OneDimLinearMeshGen(0,1,3);

elemat1 = ReactionElemMatrix(lambda,eID,msh);

elemat2 = [ 1/9 1/18 ; 1/18 1/9];
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)

```

### ElemVector function

```

function f = ElemVector(F,eN,mesh)
% function to calculate local element matrix for diffusion term

x0 = mesh.elem(eN).x(1);
x1 = mesh.elem(eN).x(2);
J = mesh.elem(eN).J;

switch length(F)
    case 1
        f = [F;F] * J;
    case 2
        A = [F(1);F(1)] * J;
        B = [(2*x0 + x1)*F(2)/3 ; (x0 + 2*x1)*F(2)/3] * J;
        f = A + B;
    otherwise
        error('Invalid source term')
end

end

```

## FEMsolver function

```

function [x,c] = FEMsolver(xmin,xmax,Ne,D,lambda,F,BCs)
% FEM solver.

% initialise mesh
mesh = OneDimLinearMeshGen(xmin, xmax, Ne);

% create and initialise global matrix and vector
M = zeros(mesh.ngn, mesh.ngn);
f = zeros(mesh.ngn,1);

%loop over elements
for i = 1:Ne
    % calculate local element matrices and add to global matrix
    L = LaplaceElemMatrix(D,i,mesh);
    R = ReactionElemMatrix(lambda,i,mesh);
    M(i:i+1, i:i+1) = M(i:i+1, i:i+1) + L - R;
    % calculate local element vectors and add to global vector
    f(i:i+1) = f(i:i+1) + ElemVector(F,i,mesh);
end

% apply boundary conditions

% first, neumann
if ~isnan(BCs(3)) % at xmin
    f(1) = f(1) - BCs(3);
end
if ~isnan(BCs(4)) % at xmax
    f(mesh.ngn) = f(mesh.ngn) + BCs(4);
end

% then, dirichlet
if ~isnan(BCs(1)) % at xmin
    M(1,:) = 0;
    M(1,1) = 1;
    f(1) = BCs(1);
end
if ~isnan(BCs(2)) % at xmax
    M(mesh.ngn,:) = 0;
    M(mesh.ngn,mesh.ngn) = 1;
    f(mesh.ngn) = BCs(2);
end

% solve the final matrix system
c = M \ f;
x = mesh.nvec';
% plot the solution vector

end

```

## analytical\_tests script

```

%% Analytical Test 1ci: Laplace's Equation with two Dirichlet BCs
xmin = 0;
xmax = 1;
Ne = 4;
D = 1;
lambda = 0;
F = 0;
BCs = [2 0 NaN NaN]; % dirichlet boundary conditions

[x,c] = FEMsolver(xmin,xmax,Ne,D,lambda,F,BCs);
xA = x; cA = 2*(1-x);

plot(x,c,'x-',xA,cA,'--');
title("Laplace's Equation with Two Dirichlet Boundary Conditions");
xlabel('x'); ylabel('c');
legend('numerical','analytical');
grid on

% save as png
saveas(gcf, 'Test1ci.png')

pause(0.1)

%% Analytical Test 1cii: Laplace's Equation with Dirichlet and Neumann
Bxmin = 0;
BCs = [NaN 0 2 NaN]; % one dirichlet and one neumann boundary condition

[x,c] = FEMsolver(xmin,xmax,Ne,D,lambda,F,BCs);

plot(x,c,'x-');
title("Laplace's Equation with Dirichlet and Neumann Boundary Conditions");
xlabel('x'); ylabel('c');
grid on

% save as png
saveas(gcf, 'Test1cii.png')

pause(0.1)

%% Analytical Test 1ciii: Diffusion-Reaction Equation
Ne = [2 3 4 5 50]; % different numbers of elements to test effect of mesh
density
lambda = -9; % reaction coefficient
BCs = [0 1 NaN NaN]; % dirichlet boundary conditions

% plot analytical solution
xA = 0:0.01:1;
cA = (exp(3) / (exp(6)-1)) * (exp(3*xA) - exp(-3*xA));
plot(xA,cA,'k','Linewidth',1);
grid on
ylim([0 1])
hold on

labels{1} = 'analytical';
for i = 1:length(Ne) % plot numerical solution with different mesh
resolutions
    [x,c] = FEMsolver(xmin,xmax,Ne(i),D,lambda,F,BCs);

```

```

    plot(x,c,'--');
    labels{i+1} = num2str(Ne(i), '%d elements');
end
legend(labels,'Location','northwest')
title("Diffusion-Reaction Equation with Different Mesh Resolutions");
xlabel('x'); ylabel('c');

% save as png
saveas(gcf, 'Test1ciii.png')

pause(0.1)
close
clearvars

```

### ThermophoresisCaseStudy script

```

% This script was written to model and simulate the behaviour of a
% thermophoresis system.

```

```

%% defining parameters
xmin = 0;
xmax = 0.01;
Ne = 500;

Tout = 323.15;
Tin = 293.15;
BCs = [Tout Tin NaN NaN];

k = 1.01e-5;

Q = 0.5:0.25:1.5;
Tl = 294.15:7:322.15;

%% modelling effects of varying flow rate Q
cQ = zeros(Ne+1,length(Q),3);
for i = 1:length(Q)
    % min Tl
    F = Q(i)*min(Tl);
    [x, cQ(:,i,1)] = FEMsolver(xmin,xmax,Ne,k,-Q(i),F,BCs);
    % mean Tl
    F = Q(i)*mean(Tl);
    [~, cQ(:,i,2)] = FEMsolver(xmin,xmax,Ne,k,-Q(i),F,BCs);
    % max Tl
    F = Q(i)*max(Tl);
    [~, cQ(:,i,3)] = FEMsolver(xmin,xmax,Ne,k,-Q(i),F,BCs);
end
% cQ(:,(length(Q)+1)/2,:) = [];

%% modelling effects of varying liquid temp Tl
cTl = zeros(Ne+1,length(Tl),3);
for i = 1:length(Tl)
    % min Q
    F = Tl(i)*min(Q);
    [~, cTl(:,i,1)] = FEMsolver(xmin,xmax,Ne,k,-min(Q),F,BCs);
    % mean Q
    F = Tl(i)*mean(Q);
    [~, cTl(:,i,2)] = FEMsolver(xmin,xmax,Ne,k,-mean(Q),F,BCs);
    % max Q
    F = Tl(i)*max(Q);
    [~, cTl(:,i,3)] = FEMsolver(xmin,xmax,Ne,k,-max(Q),F,BCs);
end

```



```

end

%% comparing linear or constant source term
cS = zeros(Ne+1,2);
a = 1; b = 4;
% min Q, Tl
F = [a*min(Q)*min(Tl) b*min(Q)*min(Tl)];
[~, cS(:,1,1)] = FEMsolver(xmin,xmax,Ne,k,-min(Q),F(1),BCs); % constant
[~, cS(:,2,1)] = FEMsolver(xmin,xmax,Ne,k,-min(Q),F,BCs); %linear
% mean Q,Tl
F = [a*mean(Q)*mean(Tl) b*mean(Q)*mean(Tl)];
[~, cS(:,1,2)] = FEMsolver(xmin,xmax,Ne,k,-mean(Q),F(1),BCs);
[~, cS(:,2,2)] = FEMsolver(xmin,xmax,Ne,k,-mean(Q),F,BCs);
% max Q, Tl
F = [a*max(Q)*max(Tl) b*max(Q)*max(Tl)];
[~, cS(:,1,3)] = FEMsolver(xmin,xmax,Ne,k,-max(Q),F(1),BCs);
[~, cS(:,2,3)] = FEMsolver(xmin,xmax,Ne,k,-max(Q),F,BCs);

%% varying source term parameters (with mean Q and Tl)
a = 1; b = [8 4 0 -4 -8];
cSp = zeros(Ne+1,length(b));

for i = 1:length(b)
    F = [a*mean(Q)*mean(Tl) b(i)*mean(Q)*mean(Tl)];
    [~, cSp(:,i)] = FEMsolver(xmin,xmax,Ne,k,-mean(Q),F,BCs);
end

% to do: -generate labels -calculate grad and curv -plot trio and save

%% generate labels for legends
% varying Q
Qlabels = string([]);
for i = 1:length(Q)
    Qlabels(i) = strcat("Q = ", num2str(Q(i)));
end

% varying Tl
Tlabels = string([]);
for i = 1:length(Tl)
    Tlabels(i) = strcat("Tl = ", num2str(Tl(i)));
end

% varying source term
Slabels = {'constant(min Q&Tl)', 'linear(minQ&Tl)',...
           'constant(mean Q&Tl)', 'linear(mean Q&Tl)',...
           'constant(max Q&Tl)', 'linear(max Q&Tl)'};

% varying source term parameters
Splabels = {};
for i = 1:length(b)
    Splabels{i} = num2str(b(i), 'f = QT_1 + %dQT_1 x');
end

%% plot results
% varying Q
figure('units','normalized','outerposition',[0 0 1 1]) % maximise figure

subplot(2,3,1); % min Tl
plot(x,cQ(:, :, 1));

```

```

grid on
title('Effects on Temperature of Varying Q (min Tl)')
xlabel('x (m)'); ylabel('T (Kelvin)');
legend(Qlabels)

subplot(2,3,2); %mean Tl
plot(x,cQ(:, :, 2));
grid on
title('Effects on Temperature of Varying Q (mean Tl)')
xlabel('x (m)'); ylabel('T (Kelvin)');
legend(Qlabels)

subplot(2,3,3); % max Tl
plot(x,cQ(:, :, 3));
grid on
title('Effects on Temperature of Varying Q (max Tl)')
xlabel('x (m)'); ylabel('T (Kelvin)');
legend(Qlabels)

% varying Tl
subplot(2,3,4); % min Q
plot(x,cTl(:, :, 1));
grid on
title('Effects on Temperature of Varying Tl (min Q)')
xlabel('x (m)'); ylabel('T (Kelvin)');
legend(Tlabels)

subplot(2,3,5); % mean Q
plot(x,cTl(:, :, 2));
grid on
title('Effects on Temperature of Varying Tl (mean Q)')
xlabel('x (m)'); ylabel('T (Kelvin)');
legend(Tlabels)

subplot(2,3,6); % max Q
plot(x,cTl(:, :, 3));
grid on
title('Effects on Temperature of Varying Tl (max Q)')
xlabel('x (m)'); ylabel('T (Kelvin)');
legend(Tlabels)

% save as png
saveas(gcf, 'parameter_space_temp.png')
pause(0.1); close

% varying source term
figure('units', 'normalized', 'outerposition', [0.25 0.25 0.5 0.5]) % resize
figure
subplot(1,1,1);
plot(x,cS(:, 1, 1), 'r:', x, cS(:, 2, 1), 'b:');
hold on
grid on
plot(x,cS(:, 1, 2), 'r-.', x, cS(:, 2, 2), 'b-.');
plot(x,cS(:, 1, 3), 'r--', x, cS(:, 2, 3), 'b--');

title('Effects on Temperature of Linear Source Term')
xlabel('x (m)'); ylabel('T (Kelvin)');
legend(Slabels, 'Location', 'southwest')

```

```

% save as png
saveas(gcf, 'source_term_temp.png')
pause(0.1); close

%% finding gradients
h = x(2)-x(1);
% varying Q
cQgrad = zeros(Ne+1,length(Q) ,3);
cQgrad2 = zeros(Ne+1,length(Q) ,3);
for i = 1:length(Q)
    for j = 1:3
        cQgrad(:,i,j) = gradient(cQ(:,i,j),h);
        cQgrad2(:,i,j) = gradient(cQgrad(:,i,j),h);
    end
end

% varying Tl
cTlgrad = zeros(Ne+1,length(Tl),3);
cTlgrad2 = zeros(Ne+1,length(Tl),3);
for i = 1:length(Tl)
    for j = 1:3
        cTlgrad(:,i,j) = gradient(cTl(:,i,j),h);
        cTlgrad2(:,i,j) = gradient(cTlgrad(:,i,j),h);
    end
end

% varying source term
cSgrad = zeros(Ne+1,2,size(cS,3));
cSgrad2 = zeros(Ne+1,2,size(cS,3));
for i = 1:2
    for j = 1:size(cS,3)
        cSgrad(:,i,j) = gradient(cS(:,i,j),h);
        cSgrad2(:,i,j) = gradient(cSgrad(:,i,j),h);
    end
end

% varying source term parameters
cSpgrad = zeros(Ne+1,length(b));
cSpgrad2 = zeros(Ne+1,length(b));
for i = 1:length(b)
    cSpgrad(:,i) = gradient(cSp(:,i),h);
    cSpgrad2(:,i) = gradient(cSpgrad(:,i),h);
end

%% plot gradients
% varying Q
figure('units','normalized','outerposition',[0 0 1 1]) % maximise figure
subplot(2,3,1); % min
plot(x,cQgrad(:,:,1));
grid on
title('Effects on Temperature Gradient of Varying Q (min Tl)')
xlabel('x (m)'); ylabel('dT/dx (Kelvin/m)');
legend(Qlabels,'Location','south')

subplot(2,3,2); %mean
plot(x,cQgrad(:,:,2));
grid on
title('Effects on Temperature Gradient of Varying Q (mean Tl)')
xlabel('x (m)'); ylabel('dT/dx (Kelvin/m)');
legend(Qlabels,'Location','south')

```

```

subplot(2,3,3); % max
plot(x,cQgrad(:,:,3));
grid on
title('Effects on Temperature Gradient of Varying Q (max Tl)')
xlabel('x (m)'); ylabel('dT/dx (Kelvin/m)');
legend(Qlabels,'Location','south')

% varying Tl
subplot(2,3,4); % min
plot(x,cTlgrad(:,:,1));
grid on
title('Effects on Temperature Gradient of Varying Tl (min Q)')
xlabel('x (m)'); ylabel('dT/dx (Kelvin/m)');
legend(Tlabels,'Location','south')

subplot(2,3,5);
plot(x,cTlgrad(:,:,2));
grid on
title('Effects on Temperature Gradient of Varying Tl (mean Q)')
xlabel('x (m)'); ylabel('dT/dx (Kelvin/m)');
legend(Tlabels,'Location','south')

subplot(2,3,6);
plot(x,cTlgrad(:,:,3));
grid on
title('Effects on Temperature Gradient of Varying Tl (max Q)')
xlabel('x (m)'); ylabel('dT/dx (Kelvin/m)');
legend(Tlabels,'Location','south')

% save as png
saveas(gcf, 'parameter_space_grad.png')
pause(0.1); close

% varying source term
figure('units','normalized','outerposition',[0.25 0.25 0.5 0.5]) % resize
figure
subplot(1,1,1);
plot(x,cSgrad(:,1,1),'r:',x,cSgrad(:,2,1),'b:');
hold on
grid on
plot(x,cSgrad(:,1,2),'r-.',x,cSgrad(:,2,2),'b-.');
plot(x,cSgrad(:,1,3),'r--',x,cSgrad(:,2,3),'b--');

title('Effects on Temperature Gradient of Linear Source Term')
xlabel('x (m)'); ylabel('dT/dx (Kelvin/m)');
legend(Slabels,'Location','south')

% save as png
saveas(gcf, 'source_term_grad.png')
pause(0.1); close

%% plot curvatures
% varying Q
figure('units','normalized','outerposition',[0 0 1 1]) % maximise figure
subplot(2,3,1); % min
plot(x,cQgrad2(:,:,1));
grid on
title('Effects on Temperature Curvature of Varying Q (min Tl)')

```

```

xlabel('x (m)'); ylabel('d^2T/dx^2 (Kelvin/m^2)');
legend(Qlabels)

subplot(2,3,2); %mean
plot(x,cQgrad2(:,:,2));
grid on
title('Effects on Temperature Curvature of Varying Q (mean Tl)')
xlabel('x (m)'); ylabel('d^2T/dx^2 (Kelvin/m^2)');
legend(Qlabels)

subplot(2,3,3); % max
plot(x,cQgrad2(:,:,3));
grid on
title('Effects on Temperature Curvature of Varying Q (max Tl)')
xlabel('x (m)'); ylabel('d^2T/dx^2 (Kelvin/m^2)');
legend(Qlabels)

% varying Tl
subplot(2,3,4); % min
plot(x,cTlgrad2(:,:,1));
grid on
title('Effects on Temperature Curvature of Varying Tl (min Q)')
xlabel('x (m)'); ylabel('d^2T/dx^2 (Kelvin/m^2)');
legend(Tlabels)

subplot(2,3,5);
plot(x,cTlgrad2(:,:,2));
grid on
title('Effects on Temperature Curvature of Varying Tl (mean Q)')
xlabel('x (m)'); ylabel('d^2T/dx^2 (Kelvin/m^2)');
legend(Tlabels)

subplot(2,3,6);
plot(x,cTlgrad2(:,:,3));
grid on
title('Effects on Temperature Curvature of Varying Tl (max Q)')
xlabel('x (m)'); ylabel('d^2T/dx^2 (Kelvin/m^2)');
legend(Tlabels)

% save as png
saveas(gcf, 'parameter_space_curv.png')
pause(0.1); close

% varying source term
figure('units','normalized','outerposition',[0.25 0.25 0.5 0.5]) % resize
figure
subplot(1,1,1);
plot(x,cSgrad2(:,1,1),'r:',x,cSgrad2(:,2,1),'b:');
hold on
grid on
plot(x,cSgrad2(:,1,2),'r-.',x,cSgrad2(:,2,2),'b-.');
plot(x,cSgrad2(:,1,3),'r--',x,cSgrad2(:,2,3),'b--');

title('Effects on Temperature Curvature of Linear Source Term')
xlabel('x (m)'); ylabel('d^2T/dx^2 (Kelvin/m^2)');
legend(Slabels, 'Location','southwest')

% save as png
saveas(gcf, 'source_term_curv.png')

```

```

pause(0.1); close

%% plotting tiled source term figures - constant/linear

% constant/linear temp
figure('units','normalized','outerposition',[0.33 0 0.33 1]) % resize
figure
subplot(3,1,1);
plot(x,cS(:,1,1),'r:',x,cS(:,2,1),'b:');
hold on
grid on
plot(x,cS(:,1,2),'r-.',x,cS(:,2,2),'b-.');
plot(x,cS(:,1,3),'r--',x,cS(:,2,3),'b--');

title('Effects on Temperature of Linear Source Term')
xlabel('x (m)'); ylabel('T (Kelvin)');
legend(Slabels, 'Location','southwest')

% constant/linear gradient
subplot(3,1,2);
plot(x,cSgrad(:,1,1),'r:',x,cSgrad(:,2,1),'b:');
hold on
grid on
plot(x,cSgrad(:,1,2),'r-.',x,cSgrad(:,2,2),'b-.');
plot(x,cSgrad(:,1,3),'r--',x,cSgrad(:,2,3),'b--');

title('Effects on Temperature Gradient of Linear Source Term')
xlabel('x (m)'); ylabel('dT/dx (Kelvin/m)');
legend(Slabels,'Location','south')

% constant/linear curvature
subplot(3,1,3);
plot(x,cSgrad2(:,1,1),'r:',x,cSgrad2(:,2,1),'b:');
hold on
grid on
plot(x,cSgrad2(:,1,2),'r-.',x,cSgrad2(:,2,2),'b-.');
plot(x,cSgrad2(:,1,3),'r--',x,cSgrad2(:,2,3),'b--');

title('Effects on Temperature Curvature of Linear Source Term')
xlabel('x (m)'); ylabel('d^2T/dx^2 (Kelvin/m^2)');
legend(Slabels, 'Location','southwest')

% print A4 size png
fig = gcf;
fig.PaperUnits = 'centimeters';
fig.PaperPosition = [0 0 20 45];
print('source_term_trio.png','-dpng','-r0')

pause(0.1); close

%% plotting tiled source term figures - parameters

% parameter temp
figure('units','normalized','outerposition',[0.33 0 0.33 1]) % resize
figure
subplot(3,1,1);
plot(x,cSp);

title('Effects on Temperature of Source Term Parameters')

```

```

xlabel('x (m)'); ylabel('T (Kelvin)');
legend(Splabels, 'Location','southwest')

% parameter gradient
subplot(3,1,2);
plot(x,cSpgrad);

title('Effects on Temperature Gradient of Source Term Parameters')
xlabel('x (m)'); ylabel('dT/dx (Kelvin/m)');
legend(Splabels, 'Location','southwest')

% parameter curvature
subplot(3,1,3);
plot(x,cSpgrad2);

title('Effects on Temperature Curvature of Source Term Parameters')
xlabel('x (m)'); ylabel('d^2T/dx^2 (Kelvin/m^2)');
legend(Splabels, 'Location','southwest')

% print A4 size png
fig = gcf;
fig.PaperUnits = 'centimeters';
fig.PaperPosition = [0 0 20 45];
print('source_term_para_trio.png','-dpng','-r0')

pause; close

%% looking at mesh resolution
Ne = [2 3 4 5 10 20];
figure('units','normalized','outerposition',[0.25 0.25 0.5 0.5]) % resize
figure
subplot(1,1,1)
for i = 1:length(Ne)
    [x,c] = FEMsolver(xmin,xmax,Ne(i),k,-max(Q),max(Q)*mean(Tl),BCs);
    plot(x,c);
    hold on
    grid on
    Mlabels{i} = num2str(Ne(i),'%d elements');
end
% plot high res for comparison
Ne = 100;
[x,c] = FEMsolver(xmin,xmax,Ne,k,-max(Q),max(Q)*mean(Tl),BCs);
plot(x,c,'k--','Linewidth',1);
labels{i+1} = num2str(Ne,'%d elements');

legend(Mlabels);
title('Investigating Effects of Mesh Density for mean Tl and max Q')
xlabel('x(m)'); ylabel('T(Kelvin)');

% save as png
saveas(gcf, 'convergence.png')
pause(0.1); close; clearvars

```