Mathematically modelling the heat transfer through a space shuttle heat shield using Fourier's equation

Modelling Techniques MatLab Assignment

Candidate Number: 10079

Summary

A number of MatLab functions were designed to the criteria of a detailed programme brief to numerically model the heat flow through a space shuttle's insulation tile. Four main simulation methods were used to mathematically simulate Fourier's heat flow equation.

The programme was then used to explore a number of simulation objectives which included looking at the effectiveness of each simulation method, the effect of tile thickness on heat flow as well as different image analysis options.

The basic programme showed that the Crank-Nicolson simulation was the best for this application, with the smallest inaccuracies ($O(\Delta x^2, \Delta t^2)$), unconditional stability and the least divergence from the true temperature answer. Additionally, an increase of thickness was shown to decrease the temperature of the tile's interior but with less and less effect at larger thicknesses. An optimum timestep of 7.5s was selected to be compatible and reliable with all four simulation methods.

Limitations in the basic programme were established in the underlying assumptions, input data and programme usefulness and robustness. Enhancements were made to improve the quality of input data, improve the programme through error checking, increase accessibility to users and create a useful shooting method design tool to choose the optimum tile thickness at a certain point. A full scale GUI and improvements to the underlying simulation methods were considered but were deemed to be impractical.

University of Bath Mechanical Engineering
April 2016

Table of Contents

Introduction and Objectives

"The tiles on a space shuttle form a crucial heat shield to protect the structure from extreme temperatures during re-entry into the atmosphere. Damage to the tiles in a critical area led to the Columbia space shuttle disaster on February 1 2003." (Johnston, 2016)

A simulation was to be created that allowed for an exploration into how heat flows through these tiles as well as a tool to assess different mathematical modelling options when researching and designing for situations like

The simulation objectives were as follows:

- Predict the temperature through the thickness of a tile during a period of a space shuttle's re-entry.
- Explore the stability, accuracy and complexity of 4 different simulation methods.
- Explore different options for analysing .jpeg files from MatLab.
- Explore the effect of tile thickness on the insulation properties.

Basic Programme Brief

The mathematical model was to be constructed in MatLab by:

1) Analysing the temperature-time graph of the space shuttle tile #597's outer surface during re-entry to the atmosphere.
2) Assuming a 1D model of tile #597.
3) Simulating the heat distribution through the tile using Fourier's equation through four different methods numerical methods:
    a. Forwards Differencing
    b. Backwards Differencing
    c. Dufort-Frankel
    d. Crank-Nicolson
4) Incorporating zero-heat flow Neumann boundary conditions to the insulated interior of the shuttle tile.
5) Assuming an initial tile temperature of 60 degrees F (15.6 degrees C) throughout the tile.
6) Tile parameters to be used as outlined in Table 1.

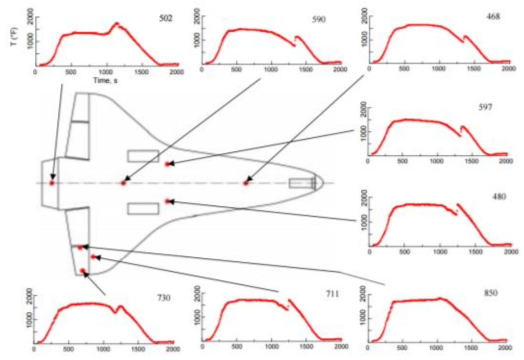


Figure 1: STS-96 windward thermocouple data from entry interface (400,000ft) to landing

Table 1: Key prescribed tile material properties required for simulation.

Tile Parameter	**Value**
Thermal Conductivity (W/mK)	0.142
Density (kg/m^3)	352
Specific Heat (J/kgK)	1256

Fourier's Equation and Simulation Methods

To model the temperature through the thickness of the space shuttle insulation tile during the period of re-entry it was necessary to use Fourier's equation, a parabolic partial differential equation modelling the variation of temperature u with time t along the length x of a uniform rod or plate.

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \qquad \alpha = \frac{k}{\rho C_p} = thermal\ diffusivity$$

This was derived by combining Fourier's law of heat conduction with an energy conservation equation (Johnston, 2016).

Fourier's equation was then adapted into 4 different numerical approximation methods (Johnston, 2016). The intricacies of each are shown in the stability testing section in Table 3.

1) Forwards
2) Dufort-Frankel
3) Backwards
4) Crank-Nicholson

Basic Programme Overview

The basic programme consisted of 4 basic functions that are outlined in Table 2 below:

Table 2: Each basic function and a brief explanation

Function	Explanation
Auto Plot Temp (Appendix A i)	Imports a .jpeg file of temperature-time space shuttle data into a user interface. The user then 'traces' the graph profile with mouse clicks to import the graph data (points, axes, origin). The data is then rescaled from pixel lengths to scaled temperature and time lengths
Shuttle Simulation 1D (Appendix A ii)	1) Defines the simulation parameters 2) Initialises simulation with one of 4 methods 3) Interpolates data to fill in missing graph information from raw data. 4) Plots a 3D surface graph of the changing temperature distribution of a 1D 'sllice' of tile when exposed to a changing heat profile.
Stability Test (Appendix A iii)	Runs through Shuttle Simulation 1D for a range of timesteps and all four simulation methods. It then extracts the final temperature value of the tile for all methods. A graph is produced which compares the final temperature results for all 4 simulation methods over the test timesteps.
Thickness Test (Appendix A iv)	Runs through Shuttle Simulation 1D for a specified range of thicknesses. Plots of the variation of the temperature of the inner wall are then plotted against time. It allows for a visualisation of the effect of thickness on heat transfer.

Temperature Plotting and Shuttle Simulation

Visuals from autoPlotTemp.m and shuttleSimulation1D.m are shown below in Figure 2 and Figure 3.

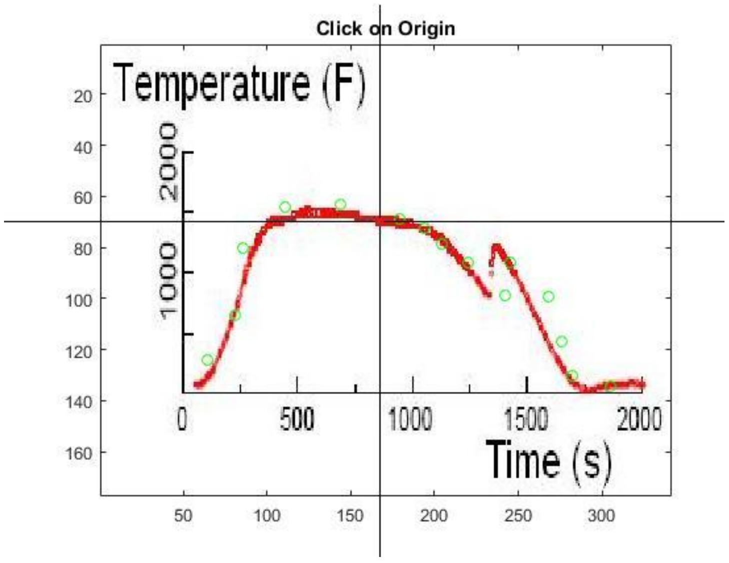


Figure 2: Manual Data entry with the basic image analyser

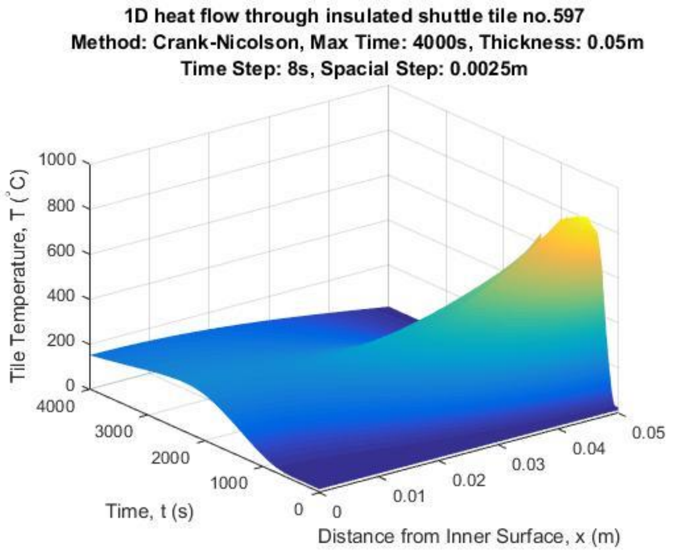


Figure 3: A typical simulation Result surface plot

Figure 2 is taken mid data collection and shows the inaccuracies of manual plotting (the green circles don't fully represent the graph).

Figure 3 shows the effect of the temperature input along the 1D slice of the tile. As would be expected, the temperature of the inner boundary of the tile raises as the shuttle re-enters atmosphere. Additionally, the Neumann boundary zero flow insulation condition is shown clearly by the non- constant interior boundary.

Stability Testing

A main objective of the simulation was to assess the efficiencies and shortcomings of each simulation method. This objective was achieved using the stabilityTest.m function (see Table 2 for detail). Figure 4 shows a visualisation of the stability of each method over a range of the Table 3 summarises the findings from the simulation testing.

A timing test was then run on shuttleSimulation1D.m using each of the different methods and taking an average speed over 4 runs. This allowed for comparison between the methods as well as quantitive data on the increase in time caused by increasing the number of timesteps.

Table 3: Information on the 4 different simulation methods used.

	Complexity Ranking (1-least, 4 – most)	**Accuracy**	**Stability**	**Comments on accuracy**	**Time to run shuttleSimulation1D.m (s)**		
					nt=501	**nt=1001**	**nt=2001**
Forwards	1	$O(\Delta x^2, \Delta t)$	$\lvert 1-4p \rvert < 1$	Drops off to infinity	0.210	0.28275	0.42125
Dufort-Frankel	2	$O(\Delta x^2, \Delta t^2)$	Unconditional	Oscillatory for large time steps	0.211	0.27925	0.426
Backwards	3	$O(\Delta x^2, \Delta t)$	Unconditional	Drops off gradually	0.221	0.3	0.4515
Crank Nicholson	4	$O(\Delta x^2, \Delta t^2)$	Unconditional	Flattest over the test timesteps	0.217	0.30675	0.46525

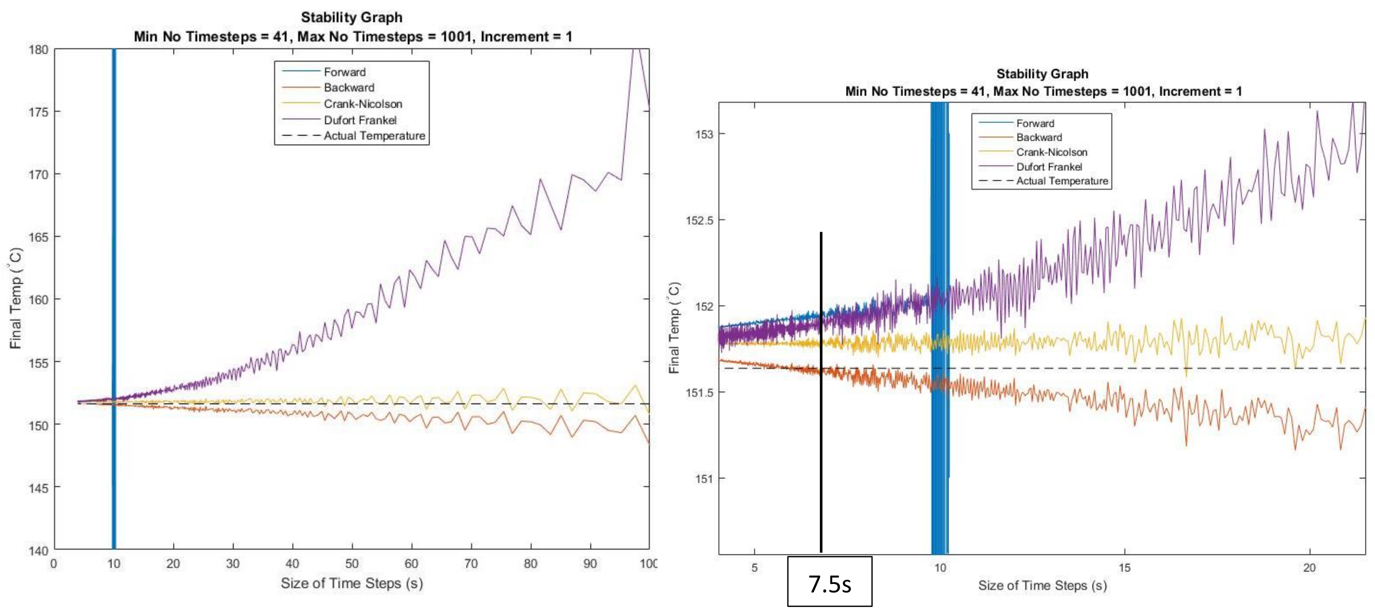


Figure 4: Graph showing the stability of four different simulation methods of a heat flow through a space shuttle heat shield.

Stability Conclusions:

- The computing time difference between the 4 different methods is very small.
- The time reductions caused by decreasing the amount of time steps is the biggest time optimisation option available.
- Therefore Crank Nicholson is best because it allows for a small number of time steps with little knock on effect for accuracy. (Important if the model were to be scaled up into simulations that require extensive processing. Eg. 2D.)
- 7.5s was deemed an appropriate time step size, as at this point, all methods are stable and there is little divergence of points. (Figure 4)

Thickness Testing

The function thicknessTest.m (see Table 2) was used to investigate the effect of tile thickness. The comparison of different interior tile temperatures are shown below in Figure 5.

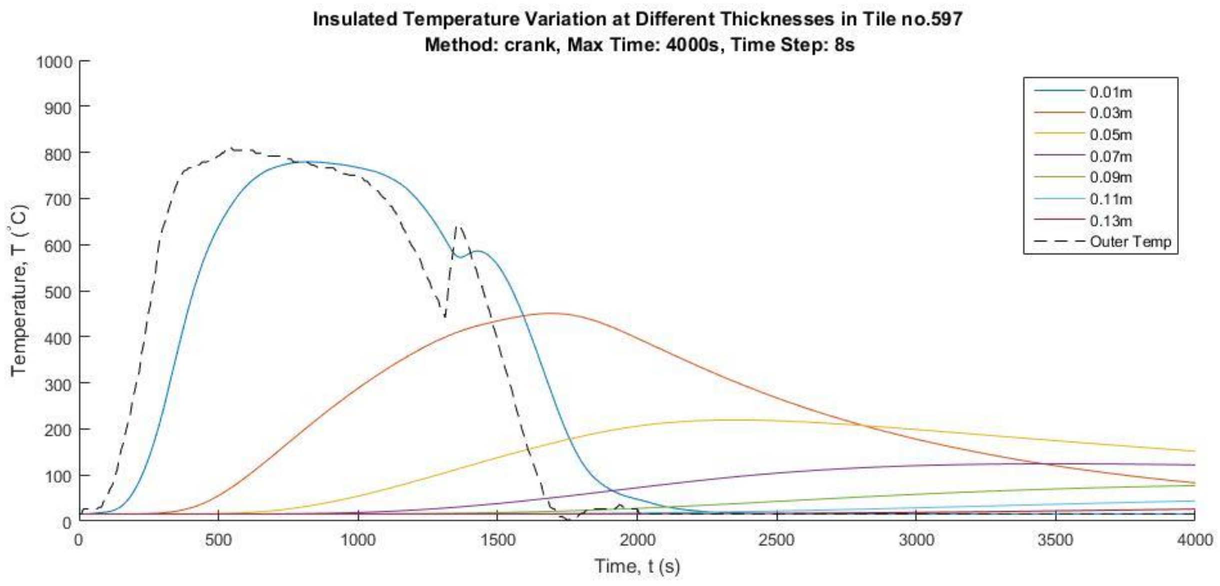


Figure 5: Thickness Testing Graph

As the thickness increased, the interior of the tile's temperature profile transitioned from being identical to the outer surface's distribution to a flat line. Initially, the temperature drops at the insulated end of the tile quickly. However, at larger thicknesses, further increase in tile thickness resulted in less decrease in maximum temperature with each step.

Past a certain thickness (0.183m - from shooting method) the input heat would not affect the inner surface of the tile at all. This simulation's use was limited and was improved into a useful tool that allowed the user to find out a minimum thickness of tile given a maximum restricting temperature value (see shooting method)

Limitations and inaccuracies of the basic programme

A number of limitations and inaccuracies in the programme were established and outlined below. **Bold** writing indicates aspects that have been addressed as appropriate for with programme enhancements.

Modelling Assumptions

- Conductivity is assumed constant over tile length.
- Density is constant and no tile imperfections.
- Inaccuracy of at least order t and x are induced per timestep.

Fixed Variables

- Constant landing temperature.
- Temperature inputs only from tile exterior.
- Uniform initial tile temperature.

Input Data

- **Manual entry graph analysis**
- **Input Data**
- Temperature data for
- Limited
- Inaccurate due to Human 'tracing' of graphs

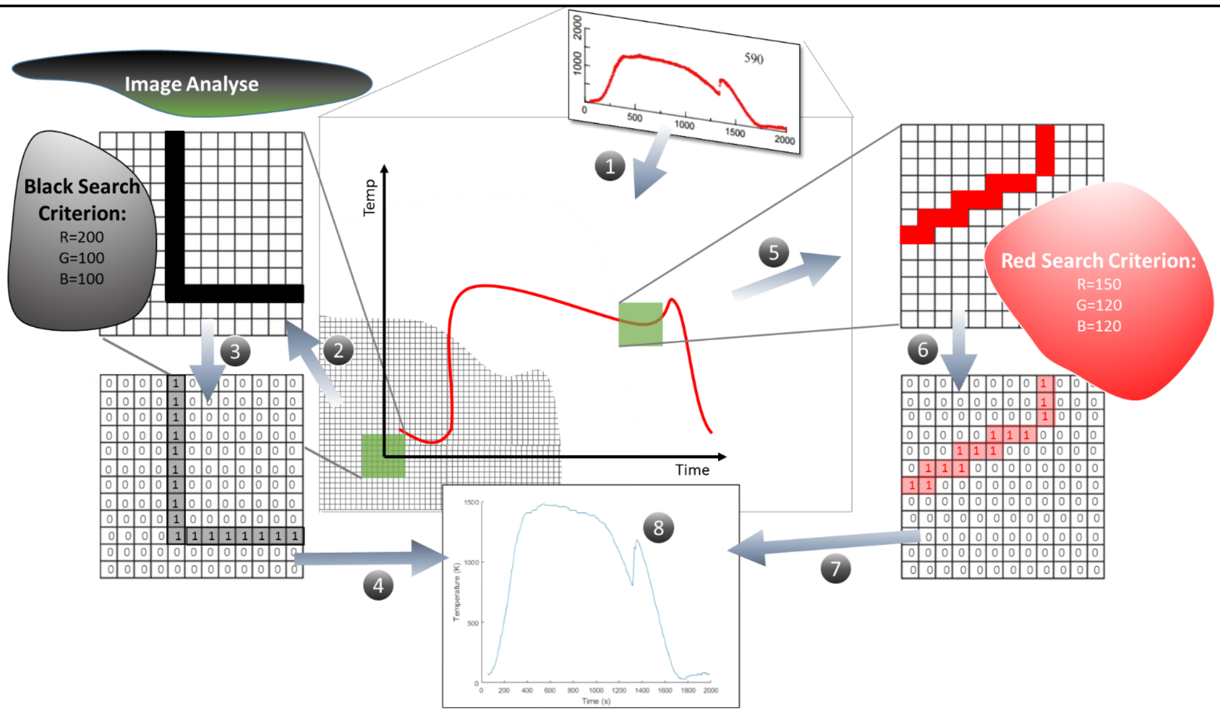
Programme

- **Speed**
- **Ease of Use**
- **False inputs**
- Limited Application
- **The thickness testing was redundant.**

Enhanced Programme

Image Analyse and Amend Data – imageAnalyse.m, amendData.m

Image analyse is an image processing function. It converts graphs in picture formats that use RGB coloured pixels into vectors of scaled numerical data that can then be processed by MatLab in further calculations involving the temperatures at different times over the space shuttle's heat shield. A visual summary is shown below in **Error! Reference source not found.**.



1) Imports the raw .jpg file of the space shuttle temperature data
2) Scans through the entire pixel grid for black pixels under the criteria (R=200, G=100, B=100)
3) Converts the pixel grid to 1s and 0s
4) Takes sums of the rows and columns of 1/0 grid. The highest value sum for the column and row pixels gives the axis location as well as the axis length.
5) Scans through the entire pixel grid for red pixels with the criteria (R=150, G=120, B=120)
6) Converts the pixel grid to 1s and 0s
7) Scans through each column of pixels. It then takes an average row number of the location of the pixels. This gives a location for the red pixels.
8) By calculating a scaling factor from the lengths of the pixel axes to the numerical length of the .jpeg graph a scale factor is used to generate to final MatLab graph.

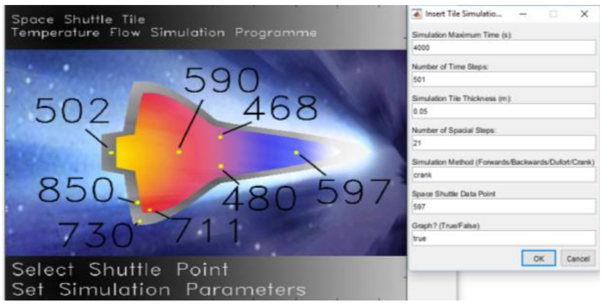
Figure 6: A visual summary of the image analyse function and associated sub processes accompanied with explanation

There were possibly some limitations to the image analyser, for example, if the graph is not landscape or there are other large lines in the image, the filtering process struggles to correctly isolate the graph data. However, the concepts could easily be scaled up and adapted to fit a wider range of application.

User Interface and Basic Error Checking: shuttleStart.m

It was decided that the benefits of programming a full graphical user interface did not justify the time investment. It was deemed far more important to diversify the functionality and improve the robustness of the underlying code. However, a basic interface was assembled that allowed for visual data entry that ran the Shuttle simulation programme (Figure 7). It did not include links to the image analyser, stability test, thickness test or the thickness shooter as these would be best included in a full scale GUI.

After data input shuttleStart would run through a number of error checking criteria to ensure reasonable simulation parameters. The programme programmed to run on a continuous loop to run the shuttle simulation as many times as the user requires.



Simulation Maximum Time (s): 4000
Number of Time Steps: 501
Simulation Tile Thickness (m): 0.05
Number of Spacial Steps: 21
Simulation Method (Forwards/Backwards/Dufort/Crank): crank
Space Shuttle Data Point: 597
Graph? (True/False): true

Figure 7: Splash Screen with data input box

```
Error using shuttleStart (line 43)
Lots of Timesteps, Simulation will be slow
```

Figure 8: An example of a custom error checking methods during the data input process

The basic interface was good for fast testing runs as well as allowing people to use the simulator without a detailed knowledge of the underlying code or MatLab.

Shooting Method for Design – thicknessShooter.m

A shooting method was constructed to calculate the minimum tile thickness required given the boundary condition of a maximumallowable tile temperature on the insulated side of the tile. This was created to make the thickness testing more accessible to a user.

1) Initial thickness guesses are made and runs shuttleSimulation.m
2) The maximum temperature at the inner surface is then compared to the target temperature to give a value for error
3) A new weighted guess for the tile thickness is made using the shooting method at the thickness
4) This loop is iterated until the tile thickness meets the temperature condition or an arbitrary number of loops are performed.

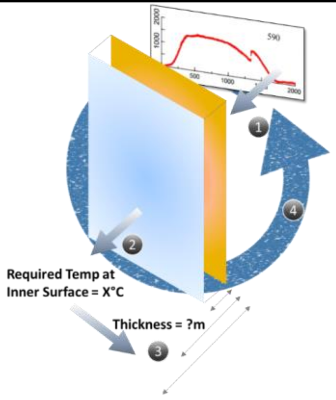


Figure 9: Visualisation and explanation of the thickness shooting method.

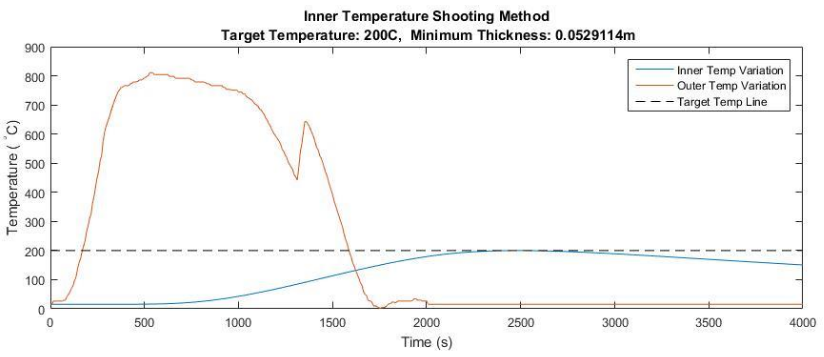


Figure 10: Shooting Method Output Plot

Conclusion

The final programme met all the objective targets and a number of clearly defined conclusion points are outlined below:

- Crank Nicolson was the best simulation method for this application, it was unconditionally stable, the least oscillatory and the most accurate for large time steps.
- The timing difference between the different methods was negligible, therefore, decreasing the amount of timesteps was the best way to speed up the programme.
- The thickness affected the internal temperature in a quadratic manner, increasing the thickness had less effect on the interior temperature distribution at higher thicknesses.
- The image analyser and shooting method have application outside this programme and could be expanded.
- It was deemed more beneficial, from an engineering perspective, to have a robust, reliable programme than a well-constructed G.U.I.

References

Johnston, D. N., 2016. *Modelling Techniques II notes,* Bath: University of Bath Mechanical Engineering Dept..

Johnston, D. N., 2016. *Shuttle Assignment Page.* [Online]
Available at: http://moodle.bath.ac.uk/mod/page/view.php?id=385449
[Accessed 22 April 2016].

Appendices:

Appendix A: Basic Programme

Commented .m files for the basic programme are shown below for reference.

Appendix A i: autoPlotTemp.m

```
% Script to plot image of measured temperature, and trace it using the mouse.
%
% Image from http://www.columbiassacrifice.com/techdocs/techreprts/AIAA_2001-0352.pdf
% Now available at
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.1075&rep=rep1&type=pdf

%Imports jpg image data into variable img
name = 'temp597';
img=imread([name '.jpg']);

%Creates Completely Blank Figure Window
figure (4);
%Plots image onto the figure
image(img);
%Keeps the image on the screen
hold on

%Uses Title As the First User Command
title('Left Click Along Curve Profile then Right Click to End Data Collecting')

%Initialise Time, Temperature and Axis Information Vectors
timeData = [];
tempData = [];
timeAxisData = [];
tempAxisData = [];
```

```
%Commence infinite loop for the manual entry of data into clicking the
%graph
while 1 % infinite loop

    % get one point using mouse
    [x, y, button] = ginput(1);

     % break if anything except the left mouse button is pressed
    if button ~= 1
        break
    end
    %Plot a green circle when the left mouse button is clicked
    plot(x, y, 'og') % 'og' means plot a green circle.

    % Add data points to vectors.
    timeData = [timeData, x];
    tempData = [tempData, y];
end

%Collect Data Calibration Data

title('Click on Origin');
[x, y, button] = ginput(1);
plot(x, y, 'og');
timeAxisData = [timeAxisData,x];
tempAxisData = [tempAxisData,y];

%Collect Temperature Scaling Factor
title('Click on 1000F Temp axis Point');
[x, y, button] = ginput(1);
plot(x, y, 'og');
tempAxisData = [tempAxisData,y];

%Collect Time Scaling Factor
title('Click on 500s Time axis Point');
[x, y, button] = ginput(1);
plot(x, y, 'og');
timeAxisData = [timeAxisData,x];

%Initialise Alignment variables for code clarity
timeAlign = timeAxisData(1);
tempAlign = tempAxisData(1);

%Initialise scaling variables for code clarity
timescale= 500/(timeAxisData(2)-timeAxisData(1));
tempscale= 1000/abs((tempAxisData(2)-tempAxisData(1)));

%Realign and then Rescale the Temperature data to give correct graph values
%in farenheight
tempData= abs(tempData-tempAlign);
tempData= tempscale.*tempData;

%Realign and then Rescale the Time data to give correct graph values in
%seconds
timeData= timeData-timeAlign;
timeData= timescale.*timeData;
```

```
% sort data and remove duplicate points.
[timeData, index] = unique(timeData);
tempData = tempData(index);

%save data to .mat file with same name as image file
save(name, 'timeData', 'tempData')
hold off
```

Appendix A ii: shuttleSimulation1D.m

```
function [x t u] = shuttleSimulation1D(tmax, nt, xmax, nx, method, doplot,point)

% Function for modelling temperature distribution in 1D in a space shuttle tile
% D N Johnston  10/04/2016
% Amended by N Rupp 2016
%
% Input arguments:
% tmax   - maximum time
% nt     - number of timesteps in simulation
% xmax   - total thickness of space shuttle tile
% nx     - number of spatial steps in simulation
% method - chosen simulation solution method ('forward', 'backward' etc)
% doplot - true to plot graph; false to suppress graph.
% point  - point on the space shuttle (tile No)
%
% Return arguments:
% x      - distance vector
% t      - time vector
% u      - temperature matrix
%
% For example, to perform a  simulation with 501 time steps
%   [x, t, u] = shuttle(4000, 501, 0.05, 21, 'forward', true, 597);

%Defines the space shuttle tile's  properties
thermcon    = 0.142;   % Thermal Conductivity W/(m K)
density     = 352;     % Density converted from 22 lb/ft^3 to (kg/m^3)
specheat    = 1256;    % 0.3 Btu/lb/F at 500F

%Calculates the value for alpha used in the Fourier equation
alpha = thermcon /(density * specheat);

%Load the tile property's raw data from an analysed Data file
fileName= sprintf('%d.mat',point');
load (fileName)

%CALCULATE VALUES FROM RAW DATA
timeMax = tmax;           %Extracts the total time of the raw data
dt = timeMax / (nt-1);   % Calculates the timestep dt
t  = (0:nt-1)*dt;        %generates a time vector
dx = xmax/(nx-1);        %Calculates the distance step by dividing total distance by no
steps
x  = (0:nx-1)*dx;        %generates a distance vector
u  = zeros(nt, nx);      %Generates an empty matrix which looks at each time and temp step
p  = alpha * dt / dx^2;  %Calculates the p value
ivec = 2:nx-1;           % Creates a vector of indices for internal points

%Set initial conditions to 60F throughout (converted to celsius)
u([1 2], :) = (60-32)*5/9;

% Begin the Main timestepping loop to model the heat flow through a tile
% using various simulation methods
for n = 2:nt - 1
    if t(n)<2000
      % Linearly interplolate outer surface data where required
        tempf = interp1(timeData, tempData, t(n+1), 'linear', 'extrap');
    else
```

```
        % To prevent interpolation after the end of the data set the tile
        % exterior to 60F
        tempf = 60;
    end
        % Convert from degrees F to degrees C
        R = (tempf - 32) * 5 / 9;
        % Set as outer surface RHS boundary condition
        u(n+1, nx) = R;

    % Select method for simulation.
    switch method
        case 'forwards'



            % LHS Neumann boundary condition to represent zero heat flow in
            % the interior of the rocket
            u(n+1, 1) = (1-2*p)*u(n,1)+ 2*p*u(n,2);
            % Internal points
            ivec=2:nx-1;
            u(n+1, ivec) = (1 - 2 * p) * u(n, ivec) + ...
                p * (u(n, ivec-1) + u(n, ivec+1));

            % RHS Dirichlet boundary condition of the exterior surface
            u(n+1, nx) = R;



        case 'dufort'
            % LHS Neumann boundary condition to represent zero heat flow in
            % the interior of the rocket
            u(n+1, 1) = ((1-2*p)*u(n-1,1)+ 4*p*u(n,2))/(1+2*p);
            % Internal points
            ivec=2:nx-1;
            u(n+1, ivec) = ((1 - 2 * p) * u(n-1, ivec) + ...
                2* p * (u(n, ivec-1) + u(n, ivec+1)))/(1+2*p);
            % RHS Dirichlet boundary condition
            u(n+1, nx) = R;

        case 'backwards'
            %Backwards Differencing Method

            %Initalise the Initial Vlues of the backwards differencing
            %matrix
            % [b c        | [u|   [d|
            % |a b c      | |u|   |d|
            % |   - - -   | |-| = |-|
            % |      a b c] |u]   |d]

            %Initial Values of b,c,d in first row
            b(1)      = 1+2*p;
            c(1)      = -2*p;
            d(1)      = u(n,1); %LHS Neumann Boundary Condition

            %General Formulae for subsequent values of
            a(2:nx-1) = -p;
            b(2:nx-1) = 1 + 2*p;
            c(2:nx-1) = -p;
            d(2:nx-1) = u(n, 2:nx-1);
            %Final and fixed values for a,b,d in the last row
            a(nx)     = 0;
            b(nx)     = 1;
            d(nx)     = R; %RHS Boundary Condition

            %Tri-Diagonal Matrix Method is then called to solve the matirx for the
            %temperature vector u (located at the bottom of the code)
            u(n+1,:) = tdm(a,b,c,d);

        case 'crank'
            %Crank Nicholson Method
            %Approximation based on Fourier's equation at the mid-point
            %between two timesteps, (x_i, t_n+delta_t/2)

            %Second order accuracy in time and space, therefore more
            %accurate but more complicated then backwards differencing,
            %therefore more computation time.

            %Initalise the Initial Values of the backwards differencing
            %matrix in the form:
            % [b c        | [u|   [d|
            % | a b c     | |u|   |d|
            % |   - - -   | |-| = |-|
            % |     a b c] |u]   |d]

            %Initial Values of b,c,d in first row
            b(1)      = 1+p;
            c(1)      = -p;
            d(1)      = (1-p)*u(n,1) + p*u(n,2); %LHS Neumann Boundary Condition

            %General Formulae for subsequant values of
            a(2:nx-1) = -p/2;
            b(2:nx-1) = 1 + p;
            c(2:nx-1) = -p/2;
            d(2:nx-1) =(p/2)*u(n,1:nx-2)+(1-p)*u(n,2:nx-1)+(p/2)*u(n,3:nx);

            %Final and fixed values for a,b,d in the last row
            a(nx)     = 0;
            b(nx)     = 1;
            d(nx)     = R; %RHS boundary condition

            %Tri-Diagonal Matrix Method is then called to solve the matrix for the
            %temperature vector u (Located at bottom of code)
            u(n+1,:) = tdm(a,b,c,d);

        otherwise
            %Error catching on incorrect method entries
            error (['Undefined method: ' method])
            return
    end
end

%Begin Plotting
if doplot

    %Find out pixel size of the screen
    scrsz = get(groot,'ScreenSize');
    %Initialise a figure in the top left of the user's screen
    figure('OuterPosition',[1 scrsz(4)/2 scrsz(3)/2.5 scrsz(4)/2])

    %Generate a surface plot of x,t,u from the simulation
    surf(x, t, u);
    shading interp %Shading parameters

    %Switch statement to extract Method name for graph title
    switch method
        case 'forwards'
            method= 'Forwards Differencing';
        case 'dufort'
            method= 'Dufort-Frankel';
        case 'backwards'
            method= 'Backwards Differencing';
        case 'crank'
            method= 'Crank-Nicolson';
    end

    %Generate graph title from simulation variables using sprintf
    graphTitle = sprintf('1D heat flow through insulated shuttle tile no.%d\n Method: %s,
Max Time: %ds, Thickness: %gm\nTime Step: %gs, Spacial Step:
%gm',point,method,tmax,xmax,dt,dx);
    title(graphTitle);

    %Label the Axes
    xlabel('Distance from Inner Surface, x (m)');
    ylabel('Time, t (s)');
    zlabel('Tile Temperature, T (^\circC)');


    %Open up a message box that, when clicked, closes the graph and returns
    %to main menu
    uiwait(msgbox('Click to Return to Parameter Menu'))
    close
    shuttleStart
end
% End of shuttle function



% ==========================================================================
% Tri-diagonal matrix solution
function x = tdm(a,b,c,d)
n = length(d);

% Eliminate a from the matrix terms
for i = 2:n
    factor = a(i) / b(i-1);
    b(i) = b(i) - factor * c(i-1);
    d(i) = d(i) - factor * d(i-1);
end

%Solve the first term in the matrix
x(n) = d(n) / b(n);

% Loop backwards to find other x values by back-substitution
for i = n-1:-1:1
    x(i) = (d(i) - c(i) * x(i+1)) / b(i);
end
```

```
% ==========================================================================
```

Appendix A iii: stabilityTest.m

```
%Stability test is a script that runs through the shuttle simulation
%numerous times using all 4 simulation methods. Each iteration the timestep
%changes. The output graph gives insight into the effectiveness of each
%simulation as the inputted number of timesteps change.

prompt          = {'Minimum Number of Timesteps:'...
                   'Maximum Number of Timesteps:'...
                   'Timestep Number Increment (s):'};

name            = 'Stability Testing';
defaultAnswer   = {'41','1001','50'};
numlines        = 1;

%Call the prompt box
inputs = inputdlg(prompt,name,numlines,defaultAnswer);

%Initialise the parameters of the thickness test with the answers from the
%input dialogue.
minTimeStep  =str2double(inputs{1}); %Minimum amount of time steps
maxTimeStep  =str2double(inputs{2}); %Maximum amount of time steps
increment    =str2double(inputs{3}); %test increment value

%Set the Shuttle simulation parameters
tMax= 4000;         %maximum time
nt = 501;           %number of timesteps in simulation
xMax=0.05;          %Overall thickness of the tile
nx = 21;            %number of spatial steps in simulation
doPlot= false;      %true to plot graph; false to suppress graph.
point   = 597;      %Space shuttle tile point chosen


%initialise the programme with i=0 at the outer boundary
i=0;

%For a an increasing number of timesteps
for nt = minTimeStep:increment:maxTimeStep
    %Increment to the next step in the displacement matrix
    i=i+1;

    %Time step at point 1= maximum time/ number of steps -1
    dt(i) = tMax/(nt-1);
    %Displays the value of the things for increasing time step
    disp (['nt = ' num2str(nt) ', dt = ' num2str(dt(i)) ' s'])
    [~, ~, u] = shuttleSimulation1D(tMax, nt, xMax, nx, 'forwards', false, point);
    uf(i) = u(end, 1);
    [~, ~, u] = shuttleSimulation1D(tMax, nt, xMax, nx, 'backwards', false, point);
    ub(i) = u(end, 1);
    [~, ~, u] = shuttleSimulation1D(tMax, nt, xMax, nx, 'crank', false, point);
    uc(i) = u(end, 1);
    [~, ~, u] = shuttleSimulation1D(tMax, nt, xMax, nx, 'dufort', false, point);
    ud(i) = u(end, 1);
```

```
end

%Generate the data for the real temperature line on the graph to act as a
%comparison point.
testSize = size (ud);
testConstant = uc(1,3);
testLine(1:testSize(2))= testConstant;

%Plot Graph
hold off %Clear previous graph

plot(dt, [uf; ub; uc; ud]); %Plot the test results
hold on                     %Prevent graph clearing
plot(dt, testLine, 'k--');  %Plot the real temperature value


%initialise and call the graph title using sprintf
graphTitle=sprintf('Stability Graph\n Min No Timesteps = %d, Max No Timesteps = %d,
Increment = %d',minTimeStep,maxTimeStep,increment);
title(graphTitle)

%Label and Scale Axes
xlim('auto')
ylim([140 180])
xlabel('Size of Time Steps (s)');
ylabel('Final Temp (^\circC)');

%Locate Legend in the Top Middle
legend ('Forward','Backward','Crank-Nicolson','Dufort Frankel','Actual
Temperature','location','north')
hold off
```

Appendix A iv: thicknessTest.m

```
function minThickness = thicknessTest(startThickness,endThickness,step)
%The function thickness test investigates the effect of thickness on the
%temperature of the internal surface of an insulated heat shield.



%Set Up Parameters of the Prompt Box
prompt          = {'Start Thickness (m):'...
                   'End Thickness (m):'...
                   'Thickness Size Increment (m):'};

name            = 'Thickness Testing';
defaultAnswer   = {'0.01','0.13','0.02'};
numlines        = 1;

%Call the prompt box
inputs = inputdlg(prompt,name,numlines,defaultAnswer);

%Initialise the parameters of the thickness test with the answers from the
%input dialogue.
startThickness  =str2double(inputs{1}); %Smallest test value of thickness
```

```
endThickness    =str2double(inputs{2}); %Largest test value of thickness
step            =str2double(inputs{3}); %test increment value

%Initialise constant variables to be used in the 1D Shuttle Simulation
%function during the testing
tMax    = 4000;       %maximum time
nt      = 501;        %number of timesteps in simulation
nx      = 21;         %number of spatial steps in simulation
method  = 'crank';    %chosen simulation solution method ('forward', 'backward' etc)
doPlot  = false;      %true to plot graph; false to suppress graph.
point   = 597;        %Space shuttle tile point chosen


%Calculate graph parameters for plotting purposes
dt = tMax / (nt-1);
noSteps= floor((endThickness-startThickness)/step);

%Initialise basic graph parameters for looped plotting
figure(1);
xlim([0 4000])

%Intialise Iterator for vector indexing purposes
i=1;
hold off % Prevent old graphs remaining on the figure

for testThickness= startThickness:step:endThickness
    hold on %Retain Graph eatch loop

    %Call Shuttle function for the current test thickness
    [~, t, u] = shuttleSimulation1D(tMax, nt, testThickness, 21, 'crank', false, 597);

    %Plot the tile interior's temperature distribution and save the current
    %thickness to a variable h for the legend.
    h(i) = plot (t,u(:,1),'DisplayName',sprintf('%gm',testThickness));

    %Prevents unneccessary calculations if the temperature no longer has an
    %effect on the interior temperature.
    if u(1,1)==u(end,1)
        minThickness=testThickness;
        break
    end
    i=i+1;
end

%Plot the original temperature on the tile's exterior as a comparison point
outerU=u(:,end);
h(end+1)= plot(t,outerU,'k--','DisplayName','Outer Temp');

%Prepare and initialise the graph title
graphTitle = sprintf('Insulated Temperature Variation at Different Thicknesses in Tile
no.%d\n Method: %s, Max Time: %ds, Time Step: %gs',point,method,tMax,dt);
title(graphTitle);

%Initialise Legend and scale and label axes
legend(h)
xlabel('Time, t (s)')
ylabel('Temperature, T (^\circC)')
ylim([0 1000]);
```

```

%Prevent future graphs from being affected by this graph
hold off

end
```

Appendix B: Additional Programmes

Commented .m files for the additional programmes are shown below for reference.

Appendix B i: imageAnalyse.m

```
function usableData = imageAnalyse(fileName, fileType)
%Image Analyse Function
%Imports the image that is to be analysed.
%This function is an image reading function that was designed to convert a
%.jpg picture of a graph with red data points into scaled data points for use in MATLAB
%This version is specifically adapted to read .jpg
%
%It:
%   1) Imports image and gets information about the size of the JPEG
%   2) Extracts the relative location of each red data point, the graphs
%   origin and axis length
%   3) Scales down the relative lengths to give correctly scaled values of temperature
%   and time
%   4) Stores the extracted data in a .m file.
%
%Inputs:
%   fileName: The name of the graph's image.
%   fileType: The type of file.
%Outputs:
%   .m file containing the extracted data for use in other functions

%DUMMY VALUES FOR TESTING
fileName='temp597';
fileType='.jpg';

%Extract all the image information from the function inputs
img=imread([fileName fileType]);

%Analyse the image to extract the image dimensions
imageSize   =size(img);
numberRows  =imageSize(1);
numberCols  =imageSize(2);

%Find the Coordinates of the Red Data Points
    %RGB Values for the target colours for the red data points
    r=150;
    g=120;
    b=120;

    %Begin a for loop to find all the red data points
    %It loops through every pixel column looking for red points. When it finds
    %multiple red point it takes an average of the point's position and stores
    %the information in rawGraphData.
    for i=1:numberCols
        rawGraphData(i)=mean(find(img(:,i,1)>r & img(:,i,2)<g & img(:,i,3)<b));
    end

%Find the Location of the Axes
    %RGB Values for the black axis points search critera
    r=200;
    g=100;
    b=100;

    %Initialise a zeros matrix for data information to be inputted into
    y= zeros(numberRows,numberCols);

    %Begin the for loop to find the location of all black pixels in the picture
    %It generates a matrix the same dimension as the imported picture. Each
    %Each pixel in  the image is then assigned a value:
    %as 1 (black) or 0 (Not black)
    %One limitation is that this for loop assumes a landscape graph
    for i=1:numberCols
        if i<=numberRows
            y(:,i)=(img(:,i,1)<r & img(:,i,2)<g & img(:,i,3)<b);
            ySumVertical(:,i)=sum(y(:,i));
            ySumHorizontal(i,:)=sum(y(i,:));
        else
            y(:,i)=(img(:,i,1)<r & img(:,i,2)<g & img(:,i,3)<b);
            ySumVertical(:,i)=sum(y(:,i));

        end
    end

    %Find coordinates of the Origin
    %Looks for the row and column where there are the most black pixels in a
    %line in the picture.
    [xBlackPoints originX] = max(ySumVertical(:));
    [yBlackPoints originY] = max(ySumHorizontal(:));

%Find the extent of the x axis
    %Begin the iteration on the right y coordinate from the origin

    %Initialise values for the while loop
    i=originX;
    test=1;
    %The while loop searches for adjacent black pixels. It begins at the axis
    %origin and searches out until there are no longer black pixels, At which point it
knows the length of the axis in pixels.
    while test
        if i==numberCols
            %An error catching loop to prevent it looping past the extent of
            %the picture
            break
        else
            %Updates the test variable. if the pixel is black Test== 1,
            %non-black == 0
            test=y(originY,i);
            %Add one to the axis length
            i=i+1;
        end
    end
    %Final Value for the X-Axis Length
    xAxisEnd= i;

%Generate the length of the Y Axis in a similar manner to the X-Axis
    i=originY;
    test=1;
    %Loop from the origin
    while test
        if i==0
```

```
            break
        else
    test=y(i,originX);
    i=i-1;
        end
    end
    yAxisEnd= i;

axisData=[originX originY xAxisEnd yAxisEnd];

%Plot raw Graph data to check for consistency and axis scaling
    amendedData=amendData(rawGraphData,axisData);
    hold on
    plot(amendedData(1,:),amendedData(2,:));
    xlabel('Time (s)');
    ylabel('Temperature (K)');
    timeData=amendedData(1,:);
    tempData=amendedData(2,:);

%save data to .mat file with same name as image file
save(fileName, 'timeData', 'tempData')

end
```

Appendix B ii: amendData.m

```
function amendedData = amendData(rawData,axisData)

%Extract Variables from function input to make code more readable
originX=axisData(1);
originY=axisData(2);
maxTimeCoord=axisData(3);
maxTempCoord=axisData(4);


%Readings from the maximum increments of the graphs
maxTimeValue=2000; % Maximum time axis value in seconds (s)
maxTempValue=2000; % Maximum temp axis value in farenheight (F)


%Find out the absolute pixel lengths of the time and temp axes
timeAxisLength=abs(maxTimeCoord-originX);
tempAxisLength=abs(maxTempCoord-originY);

%Calculate the Scale Factors between pixel length and actual length
timeScaleFactor=maxTimeValue/timeAxisLength;
tempScaleFactor=maxTempValue/tempAxisLength;

%Create and scale time vector
timeVector=[0:timeAxisLength];
timeAmend=timeScaleFactor.*timeVector;

%Create and scale temp vector
tempVector=rawData(originX:(originX+timeAxisLength));
tempVector=originY-tempVector;
```

```
tempAmend=tempScaleFactor.*tempVector;

%Clears out the NaN values from the temperature vector to prevent
%interpolation issues later on.
tempLength=length(tempAmend);
for n= tempLength:-1:1
    if isnan(tempAmend(n))
       tempAmend(n)=[];
       timeAmend(n)=[];
    end
end

% Puts the final time and temp axis data into one matrix for outputting
% from the function
amendedData(1,:)=timeAmend;
amendedData(2,:)=tempAmend;
end
```

Appendix B iii: thicknessShooter.m

```
function [designThickness] =thicknessShooter (targetTemp)
%Thickness Shooter is a function that calculates the minimum allowable
%thickness of the space shuttle given the boundary condition of maximum
%allowable temperature

%Initialise prompt box parameters
prompt          = {'What is the Maximum Allowable Temperature on the Inner Surface
(degrees C)?'}
name            = 'thicknessShooter';
defaultAnswer   = {'200'};
numlines        = 1;

%Call the prompt box
inputs = inputdlg(prompt,name,numlines,defaultAnswer);

%Initialise the parameters of the thickness test with the answers from the
%input dialogue.
targetTemp =str2double(inputs{1}); %Maximum temperature boundary condition

%Fixed Shuttle Simulation Parameters
tMax= 4000;         %maximum time
nt = 501;           %number of timesteps in simulation
nx = 50;            %number of spatial steps in simulation
method= 'crank';    %chosen simulation solution method ('forward', 'backward' etc)
doPlot= false;      %true to plot graph; false to suppress graph.
point   = 597;      %Space shuttle tile point choice

%Initial Thickness Guesses
guess1=0.1;
guess2=0.01;

%Begin Shooting Method Loop
for i=1:50

    %Call Shuttle Simulation to get the temperature matrices of each
    %simulation
    [~,~,u1] = shuttleSimulation1D(tMax, nt, guess1, nx, method, doPlot, point);
    [~,t,u2] = shuttleSimulation1D(tMax, nt, guess2, nx, method, doPlot, point);

    %Find the temperature of the insulated interior
    maxTemp(1)= max(u1(:,1));
    maxTemp(2)= max(u2(:,1));

    %Calculate the error for the current loop
    error = targetTemp-maxTemp(2);

    %Test for error each loop, if the criteria are met then break the loop
    if abs(error)<1
        break
    end

    %Call Shooting Method
    guess3 = guess2 + error*((guess2-guess1)/(maxTemp(2)-maxTemp(1)));

    %Update guesses for next guessing loop
    guess1 = guess2;
    guess2 = abs(guess3);

end

%State result from the shooting method
minThickness = guess2;

%Begin Plotting the Graph
%Variable h is used to store different legend entries
hold off
h(1)=plot(t,u2(:,1),'DisplayName','Inner Temp Variation');
hold on
h(2)=plot(t,u2(:,end),'DisplayName', 'Outer Temp Variation');

%Create a comparison line between the shooter and the final thickness value
testSize=size(t);
testLine(1:testSize(2))=targetTemp;
h(3)=plot(t,testLine,'k--','DisplayName','Target Temp Line');

%Initialise and Call Graph Title
graphTitle= sprintf('Inner Temperature Shooting Method\n Target Temperature: %dC,
Minimum Thickness: %gm',targetTemp,minThickness);
title(graphTitle)

%Initialise Legend and Axis Labelling
legend (h);
ylabel ('Temperature ( ^\circC)');
xlabel ('Time (s)');
```

Appendix B iv: shuttleStart.m

```
%shuttleStart is a script to initialise the simulation of the heat flow
%through a space shuttle tile.

%Construct the Input boxes and default values for the various tile
%simulation parameters
```

```
prompt           = {'Simulation Maximum Time (s):'...
                    'Number of Time Steps:'...
                    'Simulation Tile Thickness (m):'...
                    'Number of Spacial Steps:'...
                    'Simulation Method (Forwards/Backwards/Dufort/Crank)'...
                    'Space Shuttle Data Point'...
                    'Graph? (True/False)'};
name             = 'Insert Tile Simulation Variables';
defaultAnswer    = {'4000','501','0.05','21','crank','597','true'};
numlines         = 1;

%Bring up picture interface so that people can see the tile selection
scrsz = get(groot,'ScreenSize');        %Finds out the size of the current screen
figure('Position',[1 scrsz(4)/2 scrsz(3)/2.5 scrsz(4)/2]) %Aligns Image to the top right
of screen
img =imread('fullShuttleMerged.jpg');
image(img)

%Register the inputs from the input box
inputs = inputdlg(prompt,name,numlines,defaultAnswer);
close %Close Picture
tMax    =str2double(inputs{1});
nt      =str2double(inputs{2});
xMax    =str2double(inputs{3});
nx      =str2double(inputs{4});
method  =(inputs{5});
point   =str2double(inputs{6});
doPlot  =(inputs{7});

%Begin Basic Error Checking
if tMax < 0
    error('Please enter a positive simulation time input')

elseif nt > 4000
    error('Lots of Timesteps, Simulation will be slow')

elseif xMax >1
    error('More than 1m thick tiles? I''surprised the rocket took off in the first
place')

end

%Call the Shuttle Simulation Function
shuttleSimulation1D(tMax, nt, xMax, nx, method, doPlot, point)
```