Matlab Tile Report

11657

Juan Fran Sierra

Abstract

In this report, the temperature inside of the tile used in a space shuttle was modelled with respect to time and thickness through the tile in 1 dimension. Finite different methods were analysed to find the method with the best accuracy and stability to approximate fouriers heat equation through the tile. The Crank-Nicolson method was remained stable for the longest time and spatial steps so this was the final method used for the model. Assumptions were made to simplify the model such as assuming the model was in only one dimension instead of three or assuming their was no heat dissipated into the shuttle from the internal value. The minimum thickness required to maintain the maximum internal temperature below 175ºC with an error of 5ºC was 0.0576m.

Table of contents

Project Introduction

Computational modelling provides a more rapid and economically feasible alternative to experimental techniques when the scale of the phenomenon in question surpasses certain orders of magnitude. Forward differencing, DuFort-Frankel, Backward differencing and Crank-Nicolson are all methods that can be combined with MATLAB coding knowledge to provide a reliable approximation to the ‘real-life’ problem.

The re-entry of a space shuttle into the atmosphere will develop frictional interaction between the shuttle surface and the atmospheric environment. This will release vast amounts of heat energy that a Thermal Protection System (TPS) will have to resist. The temperature at the inner surface of the shuttle will decrease with increasing tile thicknesses. The program aims

to optimise pile thickness in order to reduce weight and cost, whilst still keeping the inner temperature of the shuttle below 175ºC [1] to protect material and personnel.

Fig.1 Space shuttle thermal protection system

The program aims to develop a visual representation of the material temperature with respect to time and space. For simplification purposes temperature is only monitored across the thickness of the material, since it is this the only dimension we are interested in. The code is assisted by Neumann boundaries and Fourier's one dimensional equation. Both assumptions such as the uniformity of the material properties and the low resolution images produced should be taken into account when using the program and assessing its reliability.

Results and Discussion

Accuracy and Stability for various methods

In this experiment, finite difference methods were used to approximate the Fourier's equation of the heat dissipation across a tile, with respect to the displacement within the tile thickness (in metres) and time (in seconds).

The function *stabilitytimestep.m* was used to find which method out of Forward, DuFort-Frankel, Backward differencing and Crank-Nicolson was the most accurate and stable for different timesteps.

The Forward and DuFort-Frankel methods are explicit methods. This means that the next value in the sequence is produced exclusively with the data from the previous value. Therefore, this makes the Forward and DuFort-Frankel methods inherently unstable[2]. The Backward differencing and Crank-Nicolson methods are implicit methods. This makes them more stable than the other methods as they use the value from the previous and next timesteps, resulting in a better approximation.

Furthermore, when referring to accuracy, the Second order approximations are typically more accurate than First order, as their error of approximation is smaller. In this case, the First order methods are the Forward and Backward method and the Second order methods the Crank-Nicolson and the DuFort-Frankel methods.

Therefore, from the analysis developed above, the method with the best accuracy and stability would be the Crank-Nicolson method. This is confirmed in Fig.2 where we can observe that the Forward method becomes extremely unstable for timesteps above 9 seconds. The Dufort-Frankel method was relatively stable until it reached around 15 seconds. The Backward approximation works accurately until it reaches a timestep of 24 seconds, were it exceeds a 0.5 ºC difference of the desired midpoint temperature.

The only method which maintains the maximum accuracy is the Crank-Nicolson, which keeps a 0.5ºC difference from the desired midpoint temperature for the longest timestep.

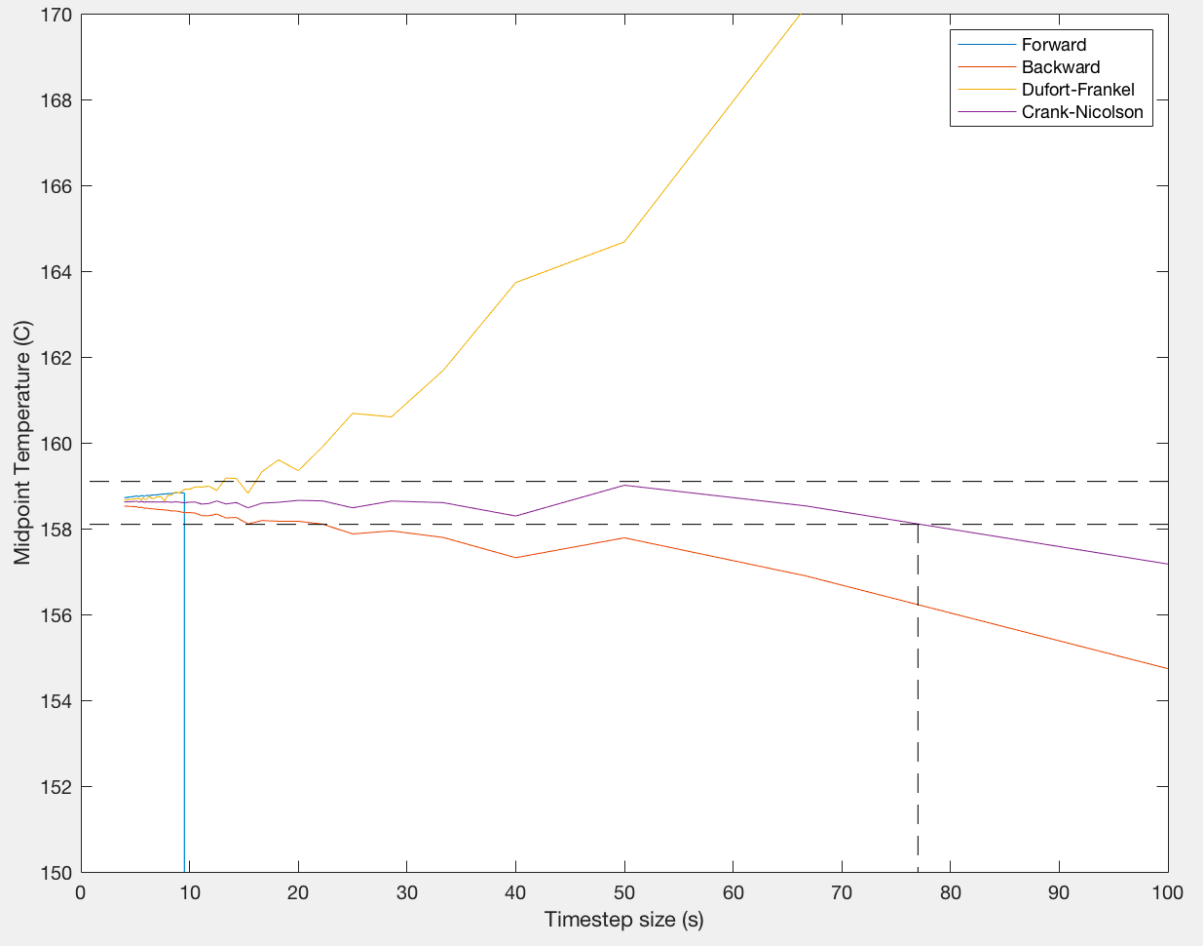

Fig.2 Tile midpoint temperature against Time step size for the 4 different methods

The timestep at which the Crank-Nicolson approximation falls below the -0.5ºC of the midpoint temperature was 77 seconds. However, a safety factor of 1.5 was used to make sure that the approximation never exceeded the temp. limit under any circumstance, even though this will require more computational value as the value for nt will be larger. Therefore, dt = $\frac{77}{1.5} \approx 51$ and nt = $\frac{4000}{51} \approx 78$. These values were used in the *shuttle.m* and *shootingmethod.m* functions.

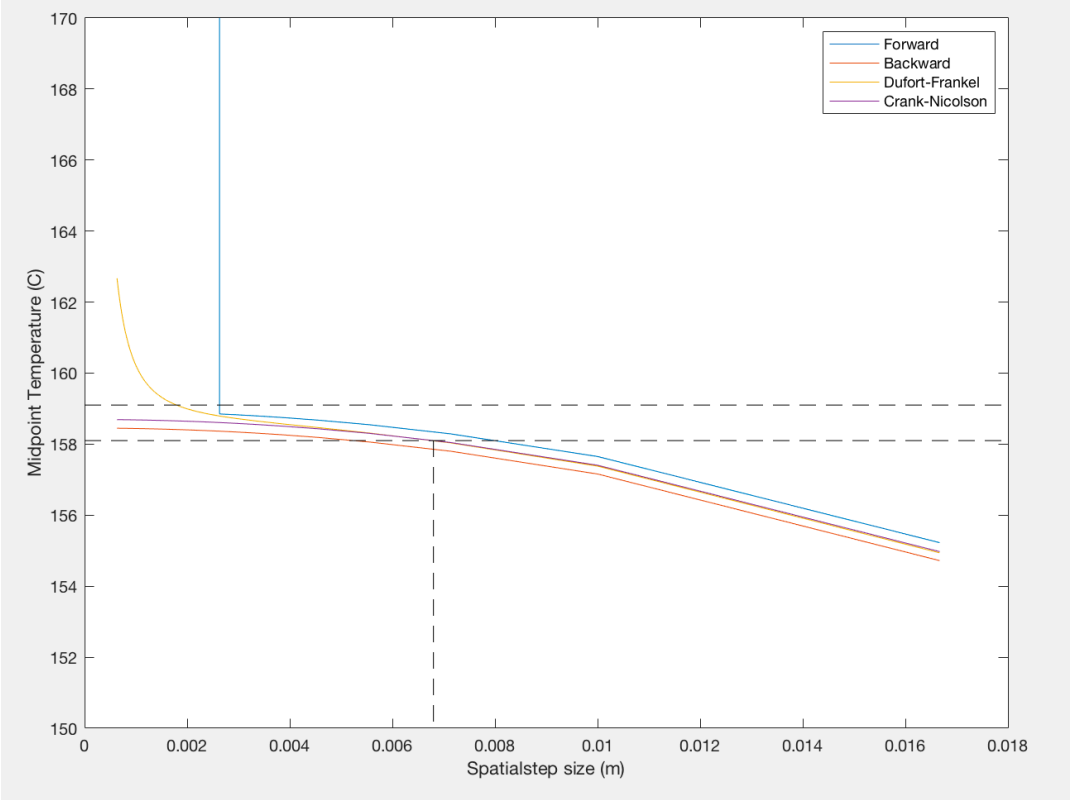

Fig.3 Tile midpoint temperature against Spatial step size for the 4 different methods

The procedure carried out to find the most suitable timestep was repeated to find the ideal spatial step. The graph in Fig.3 shows that the Forward and Dufort-Frankel method are unstable for small spatial steps (dx<0.0026). The Bakward and Crank-Nicolson remain relatively stable for all spatial step values, but the Crank-Nicolson method exceeds the -0.5°C limit when the spatial step size dx = 0.0068m. A safety factor of 1.5 is used again in case of uncertainty. Therefore, dx = $\frac{0.0068}{1.5} \approx 0.0045$m, nx = $\frac{0.05}{0.0045} \approx 11$. These values were used in the *shuttle.m* and *shootingmethod.m* functions.

Variation of temperature with time and distance through the tile

The main objective of this experiment was to identify the minimum thickness required for the tile to not exceed an inner temperature of 175°C to ensure the safety of the crew and circuit systems inside the shuttle. A simulation of the variation of temperature with time and distance through the tile was executed in matlab using the function *shuttle.m*, in which we were able to modify the thickness of the tile.

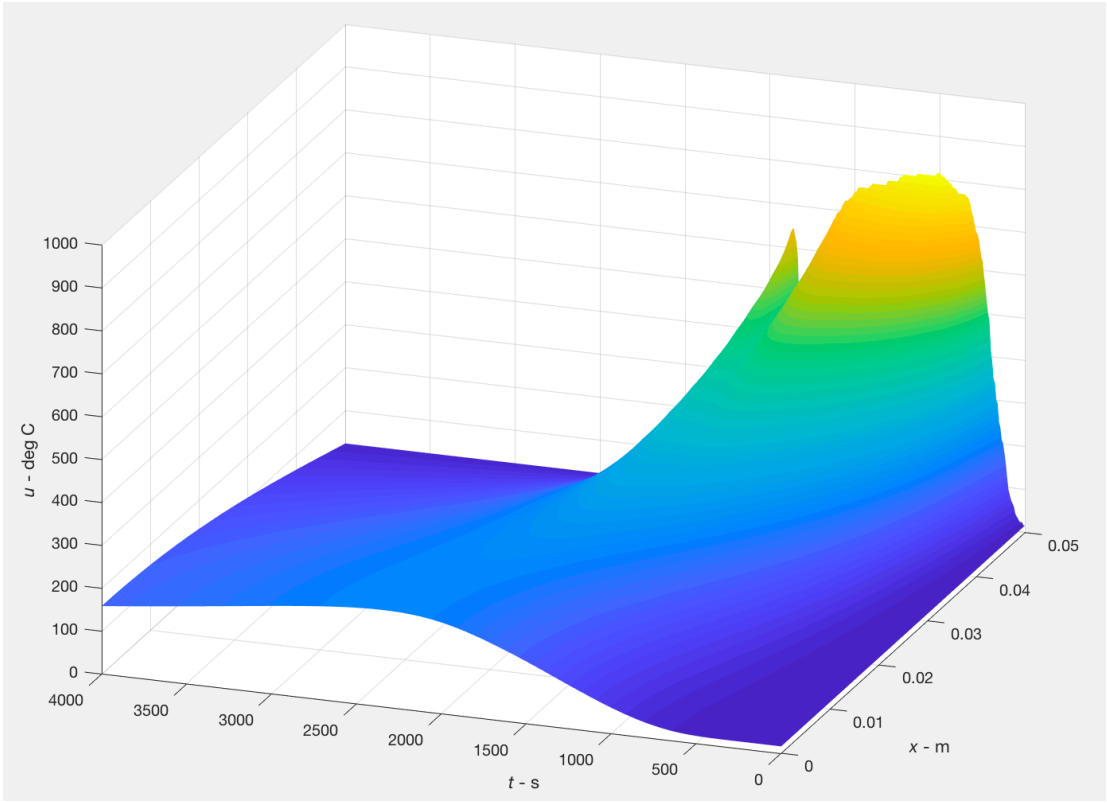

Fig.3 Variation of temperature with time and distance through a tile of 0.05m thickness

The *internaltemp.m* function was developed to plot the values of the inner temperature against time for a range of tile thicknesses to identify the minimum allowable tile thickness that was allowed without exceeding a temperature of 175ºC which could alter the material composition of the silica in the tile. The results are shown below in Fig.4.

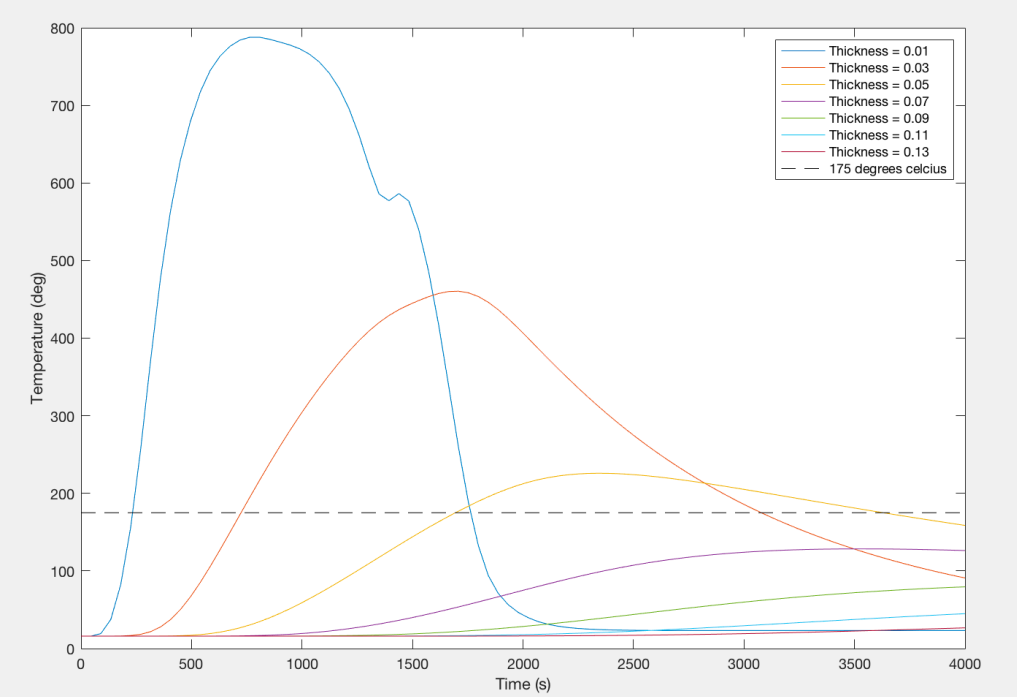

Fig.4 Graph of Inner temperature against time for different tile thicknesses

From the results in Fig.4, we can observe that the ideal tile thickness will be between 0.05 and 0.07m. In this case the minimum tile thickness required to not exceed 175ºC of inner temperature is 0.07m. Nevertheless, a better approximation is possible using an iterative method called the shooting method (shootingmethod.m) which will be later presented in the enhancements section.

Enhancements

In Fig.4 we identified that the ideal tile thickness lied between 0.05 and 0.07m. A tile value of 0.07m was chosen ,as a lack of iterations of the tile thickness lead to choosing this value in case of uncertainty in the results. However, even though using 0.07m as a tile thickness would make sure that there is no risk of surpassing the maximum inner temperature, a vast amount of unnecessary theramic (silica) would be used to make each individual tile. This would result in an obvious increase in the manufacturing costs as more raw material would be required for the tile, but also, this would enlarge the size and therefore the weight of the space shuttle, resulting in an increased cost of the fuel consumption.

To solve this problem and find the minimum tile thickness to a higher degree of accuracy, a shooting method was developed. The shooting method calculates the minimum required thickness to maintain the internal temperature of the tile below 175ºC. It makes 2 initial guesses in which the thickness is chosen at random. It then uses a while loop to reduce the distance error between the current temperature and 175ºC. Once the temperature error is less than 5ºC and below 175ºC, the while loop stops and the final thickness for this tile is used as the correct thickness for the desired maximum temperature. A simulation of this method is shown in Fig.5. The final tile thickness calculated by the *shootingmethod.m* with an error of 5ºC was 0.0576m.

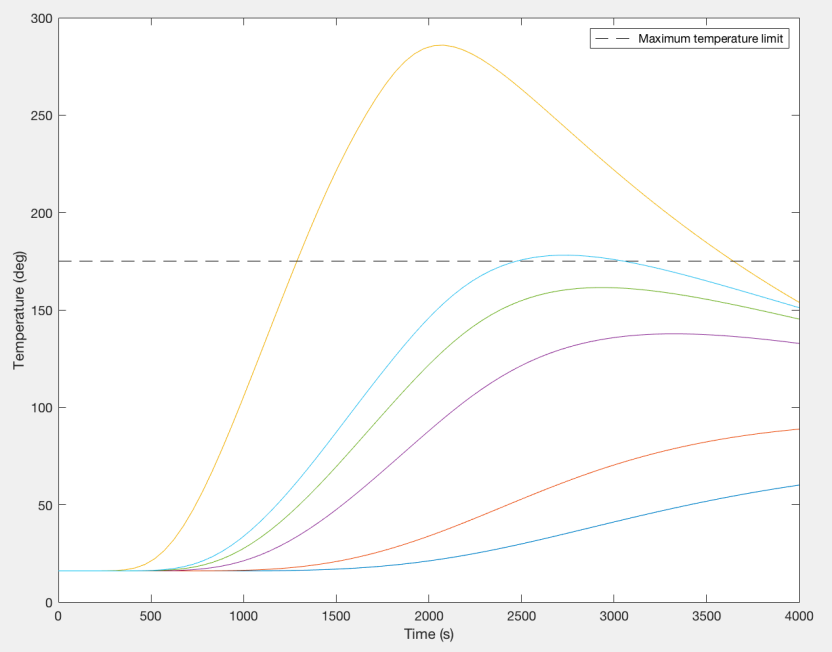

Fig.5 Shooting method iterations to find most suitable minimum thickness

Another enhancement was produced to make the data obtained from the graph shown in Fig.6 more accurate. Initially, the data had to be selected manually with the mouse. The plottemp.m program is designed to scan the pixels of the external temperature of the tile against time and extract the data required as shown in Fig.7 .

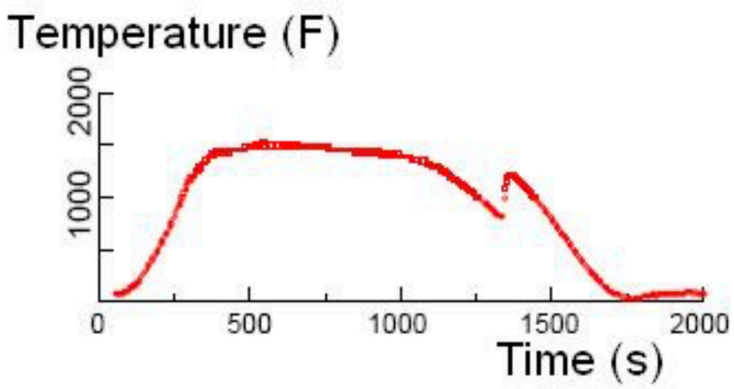

Fig.6 External temperature data source[3]

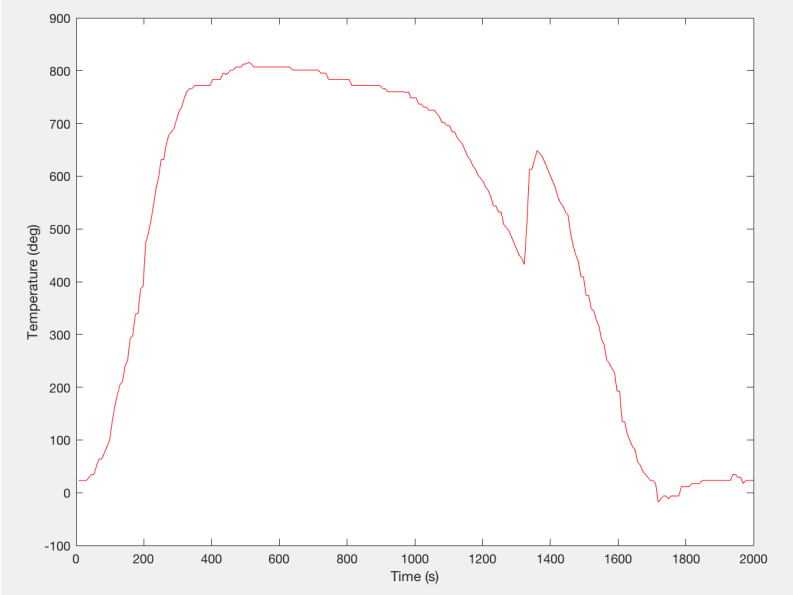

Fig.7 External temperature data extracted

Task Assumptions

This task provided a solid amount of reliable data to model the temperature through the tile. Nevertheless, there were certain assumption that made the task more simple to model and illustrate in graphs without requiring extensive computational power. It is true that to make the task more accurate and in the case of actually using the information provided in the results of this task for real life implementation in a space shuttle, the assumption should have been dealt with to make the results as accurate as possible.

Some of the assumptions in the task include:

1D thickness: The distance through the tile was modelled in one dimension. In the real tile, there are three dimensions. This would require a significant increment in the number of

equations and variables used in the model, making it harder to program and this would also require an increase in computational power.

Tile composition: The tile was assumed to be entirely made entirely of only silica. Not only this, but the density of the material throughout the tile was assumed to be exactly equal.

Neumann Boundaries: The internal surface of the tile was considered as a Neumann Boundary. In reality, this wouldn't be accurate as there is an external temperature inside the shuttle that would affect the temperature model in the tile. To make this boundary more accurate, a Robin boundary could be implemented to the model, which would take into account heat dissipation into the shuttle from the tile.

Initial Conditions: The values for density, specific heat and thermal conductivity were assumed to remain equal throughout the process. In reality, these values would change with a change in temperature.

Conclusion

In conclusion, it was found that an increase in the tile thickness resulted in a subsequent decrease of the maximum internal temperature of the tile. Nevertheless, if an unnecessarily thick tile was to be used, there would be an unnecessary expenditure in the tile material (silica) and an increase in fuel consumption due to an increment in the shuttle weight. Therefore, an optimal thickness was required in which the maximum internal temperature did not exceed 175^{o}C (with an error of 5^{o}C), but at the same time had the minimum possible thickness. Therefore, a shooting method was used to calculate the minimum thickness required to have a maximum internal temperature 5 degrees less than 175^{o}C. The optimal thickness was found to be 0.0576m.

After analysing the accuracy and stability of the different solution methods, it was concluded that the most accurate method was the Crank-Nicolson method. It maintained stability with time steps up to dt = 51s and spatial steps up to dx = 0.0045m, including a safety factor in case of uncertainty.

Refrences

[1] 'Thermal Protection Systems' Accessed: 04/04/19, Available online at: https://www.nasa.gov/centers/johnson/pdf/584728main_Wings-ch4b-pgs182-199.pdf
[2] 'Christian Grossmann; Hans-G. Roos; Martin Stynes (2007)' Accessed: 05/04/19
[3] Modelling Techniques 2 Accessed: 07/04/19, Available online at: https://moodle.bath.ac.uk/mod/page/view.php?id=385449

Appendices

Plottemp.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&t
ype=pdf
%
% D N Johnston 30/01/19name = 'temp597';

name = 'temp597';
img=imread([name '.jpg']);

for i=1:size(img,2)
  Red=img(:,i,1);
  Green= img(:,i,2);
  Blue= img(:,i,3);
  initialtempdata(i)=mean(find(Red>110 & Blue<90 & Green<70)); % Scans
pixels of image
end

%Ignore any points outside the function
initialtempdata(isnan(initialtempdata))=[];

%Flip the temperature vector
tempdataflip=max(initialtempdata)-initialtempdata;
%Convert pixels to Farenheit
tempdataF=(1500/max(tempdataflip))*tempdataflip;
%Changing to deg C
tempdata=(tempdataF-32)/1.8;

%Create time vector
time = (1:1:size(initialtempdata,2));
%Convert from pixels to seconds
finaltime = time*(2000/max(time));

% Plot graph
plot(finaltime,tempdata,'r')
xlabel('Time (s)')
ylabel('Temperature (deg)')

% Temp and time data saved
save(name, 'timedata', 'tempdata')
```

Shuttle.m

```
function [x, t, u] = shuttle(tmax, nt, xmax, nx, method, doplot)
% Function for modelling temperature in a space shuttle tile
```

```
% D N Johnston  30/1/19
%
% Input arguments:
% tmax   - maximum time
% nt     - number of timesteps
% xmax   - total thickness
% nx     - number of spatial steps
% method - solution method ('forward', 'backward' etc)
% doplot - true to plot graph; false to suppress graph.
%
% 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);
%

% Set tile properties
thermcon = 0.141; % W/(m K)
density  = 351;   % 22 lb/ft^3
specheat = 1259;  % ~0.3 Btu/lb/F at 500F
alpha = thermcon/(density * specheat);

% Some crude data to get you started:
%timedata = [0  60 500 1000 1500 1750 4000]; % s
%tempdata = [16 16 820 760  440  16   16];   % degrees C

% Better to load surface temperature data from file.
% (you need to have modified and run plottemp.m to create the file first.)
% Uncomment the following line.
load 'temp597.mat'

% Initialise everything.
dt = tmax / (nt-1);
t = (0:nt-1) * dt;
dx = xmax / (nx-1);
x = (0:nx-1) * dx;
u = zeros(nt, nx);
p = (alpha * dt) / dx^2;

% set initial conditions to 16C throughout.
% Do this for first two timesteps.
u([1 2], :) = 16;

ivec = 2:nx-1;

% Main timestepping loop.
for n = 2:nt-1

    % RHS boundary condition: outer surface.
    % Use interpolation to get temperature R at time t(n+1).
    R = interp1(timedata, tempdata, t(n+1), 'linear', 'extrap');


    % Select method.
    switch method
        case 'forward'
```

```
            % LHS Neumann boundary condition
            u(n+1,1)= (1 - 2 * p) * u(n,1) + 2 * p * u(n,2);

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

            %internal values
            u(n+1,ivec) = (1 - 2 * p) * u(n,ivec) + p * (u(n,ivec-1) +
u(n,ivec+1));


        case 'dufort-frankel'


            % LHS Neumann boundary condition
            u(n+1,1) = ((1 - 2*p)*u(n-1,1) + (4*p*u(n,2)))/(1 + 2*p);

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

            %internal values
            u(n+1,ivec) = ((1-2*p)*u(n-1,ivec) + 2*p*(u(n,ivec-1) +
u(n,ivec+1)))/(1 + 2*p);



        case 'backward'

            % LHS Neumann boundary condition;
            b(1) = 1 + 2*p;
            c(1) = -2*p;
            d(1) = u(n,1);

            % RHS Dirichlet boundary condition
            a(nx) = 0;
            b(nx) = 1;
            d(nx) = R;

             %internal values

            a(ivec) = -p;
            b(ivec) = 1 + 2*p;
            c(ivec) = -p;
            d(ivec) = u(n, ivec);

            u(n+1,:) = tdm(a,b,c,d);




        case 'crank-nicolson'

            % LHS Neumann boundary condition;
            b(1)      = 1 + p;
            c(1)      = -p;
            d(1)      = (1 - p)*u(n,1) + p*u(n,2);
```

```
            % RHS Dirichlet boundary condition
            a(nx)      = 0;
            b(nx)      = 1;
            d(nx)      = R;

            %internal values

            a(ivec) = -p/2;
            b(ivec) = 1 + p;
            c(ivec) = -p/2;
            d(ivec) = (p/2)*u(n, 1:nx-2) + (1-p)*u(n,2:nx-1) +
(p/2)*u(n,3:nx);

            u(n+1,:) = tdm(a,b,c,d);


        otherwise
            error (['Undefined method: ' method])
            return
    end
end

if doplot
    % Create a plot here.
    surf(x,t,u)
    % comment out the next line to change the surface appearance
    shading interp

    % Rotate the view


    %label the axes
    xlabel('x - m')
    ylabel('t - s')
    zlabel('Temp - deg C')
    ylim([0 4000])
    zlim([0 1000])

end
% End of shuttle function
```

Stabilitytimestep.m

```
function stabilitytimestep(tmax, nx, thick)

% Function analyses stability of the four different methods for a range of
% time steps

% Input arguments:
% tmax        - maximum time (s)
% nx          - number of spatial steps
% thick       - thickness of the tile (m)
```

```
i=0;

% Calculates the temperature values at different timesteps for each method
for nt = 41:20:1001
    i=i+1;
    dt(i) = tmax/(nt-1);
    disp (['nt = ' num2str(nt) ', dt = ' num2str(dt(i)) ' s'])
    [~, ~, u] = shuttle(tmax, nt, thick, nx, 'forward', false);
    uf(i) = u(end, 1);
    [~, ~, u] = shuttle(tmax, nt, thick, nx, 'backward', false);
    ub(i) = u(end, 1);
    [~, ~, u] = shuttle(tmax, nt, thick, nx, 'dufort-frankel', false);
    ud(i) = u(end, 1);
    [~, ~, u] = shuttle(tmax, nt, thick, nx, 'crank-nicolson', false);
    uc(i) = u(end, 1);
end

% Defines the boundary limits at which the method exceed the desired
% accuracy
boundary1 = 158.6 + 0.5;
boundary2 = 158.6 - 0.5;

% Plots the temperature againt the timestep
plot(dt, [uf; ub; ud; uc])

% Plots the boundary limits
hold on
plot([0 tmax], [boundary1 boundary1],'--k')

plot([0 tmax], [boundary2 boundary2],'--k')

plot([77 77], [0 boundary2],'--k')

% Defines axis limits and labels on graph
ylim([150 170])
xlim ([0 100])
hold off
xlabel('Timestep size (s)')
ylabel('Midpoint Temperature (C)')
legend ('Forward', 'Backward', 'Dufort-Frankel', 'Crank-Nicolson')
end
```

stabilityspatialstep.m

```
function stabilityspatialstep(tmax, dt, thick)

% Function analyses stability of the four different methods for a range of
% spatial steps

% Input arguments:
% tmax        - maximum time (s)
% dt          - timestep size
% thick       - thickness of the tile (m)

i=0;
```

```
nt = tmax/dt; % Calculates the number of spatial steps

% Calculates the temperature values at different spatialsteps for each
method
for nx = 4:1:80
    i=i+1;
    dx(i) = thick/(nx-1);
    disp (['nx = ' num2str(nx) ', dx = ' num2str(dx(i)) ' s'])
    [~, ~, u] = shuttle(tmax, nt, thick, nx, 'forward', false);
    uf(i) = u(end, 1);
    [~, ~, u] = shuttle(tmax, nt, thick, nx, 'backward', false);
    ub(i) = u(end, 1);
    [~, ~, u] = shuttle(tmax, nt, thick, nx, 'dufort-frankel', false);
    ud(i) = u(end, 1);
    [~, ~, u] = shuttle(tmax, nt, thick, nx, 'crank-nicolson', false);
    uc(i) = u(end, 1);
end

% Defines the boundary limits at which the method exceed the desired
% accuracy
boundary1 = 158.6 + 0.5;
boundary2 = 158.6 - 0.5;

% Plots the temperature againt the spatialstep
plot(dx, [uf; ub; ud; uc])

% Plots the boundary limits
hold on
plot([0 tmax], [boundary1 boundary1],'--k')

plot([0 tmax], [boundary2 boundary2],'--k')

plot([0.0068 0.0068], [0 boundary2],'--k')

% Defines axis limits and labels on graph
ylim([150 170])
xlim ([0 0.018])
xlabel('Spatialstep size (m)')
ylabel('Midpoint Temperature (C)')
legend ('Forward', 'Backward', 'Dufort-Frankel', 'Crank-Nicolson')
end
```

tdm.m

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

% Eliminate a 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

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
```

shootingmethod

```
function [suggthick,minthick]=shootingmethod(method, nx, nt, tmax,
targetInternal)
% Function calculates the minimum thickness of the tile required to
maintain the internal surface temperature below the desired limit

% Input arguments:
% method      - approximation method
% nx          - number of spatial steps
% nt          - number of timesteps
% targetInternal - internal surface temperature of tile (deg C)

% Return arguments:
% suggthick   - values of iterations for suggested thickness
% minthick     - minimum thickness to maintain a temperature below the
% desired one


    maxerror = 5;        % Defines the maximum temperature error

    suggthick = [0.01,0.1]; % To calculate the first and second errors, two
initial values for the thickness are specified


    doplot = false;         % Doesn't perform the graph plot from shuttle.m

% Finding required thickness
% Calculates inner temperature using first two thickness guesses
    for i=1:2
        [~, ~, u] = shuttle(tmax, nt, suggthick(i), nx, method, doplot);
        maxinner(i)=max(u(:,1));
        error(i) = maxinner(i) - targetInternal;
    end
 % Loop until a guess for the thickness results in an appropriately
accurate
 % inner temperature
    n = 2;
    while abs(error(end)) > maxerror  % Loop until error is smaller than
the maximum allowed
        [~, t, u] = shuttle(tmax, nt, suggthick(n), nx, method, doplot);
        error(n) = max(u(:,1)) - targetInternal;
        % Generates a linear approxiation to following guess
        G = (error(n) - error(n-1))/(suggthick(n) - suggthick(n-1)); % G =
Gradient
        I = error(n) - G*suggthick(n);                               % I =
Y Intercept
        suggthick(n+1) = -I/G;
```

```
        n = n + 1;

        %plot the different iterations
        plot([0 tmax],[targetInternal targetInternal],'--k')
        hold on
        plot(t,u(:,1))
        xlabel('Time (s)')
        ylabel('Temperature (deg)')
        legend('Maximum temperature limit')
    end
 % Extracts from the thickness array the last thickness guess which is the
final
 % solution
    minthick=suggthick(end);
end
```

internaltemp.m

```
function internalTemp(tmax,nt,nx)

% Function calculates the internal temperature against time for diffrent
% tile thicknesses

% Input arguments:
% tmax        - maximum time (s)
% nt          - number of time steps
% nx       - number of spatial steps

doplot = false;
method = 'crank-nicolson';

for xmax = 0.01:0.02:0.14 % Loop for thicknesses between 0.01 and 0.14 with
0.02m intervals
    [~, t, u] = shuttle(tmax, nt, xmax, nx, method, doplot);

    figure(1)
    plot(t,u(:,1),'DisplayName',['Thickness = ' (num2str(xmax))]) % Plots
the internal temp for each tile thcikness

    legend()
    xlabel('Time (s)')
    ylabel('Temperature (deg)')

    hold on

end

plot([0 tmax],[175 175],'--k','DisplayName', '175 degrees celcius')
hold off

end
```