Modelling of a Spacecraft Heat Shield Tile

Abstract

Space shuttles were the first reusable form of spacecraft and were successfully launched in 1981. To make a space craft reusable, one of the major design challenges to overcome was surviving the extreme temperatures whilst re-entering the atmosphere that would melt an aluminium airframe. The aim of this report is to discuss the design process of a MATLAB model of the heat experienced, during re-entry, by the tiles that formed a heat shield for the shuttle. From this model, the optimal tile thickness was calculated as **64mm**. This was the minimum tile thickness required for preventing temperature of the inner surface exceeding **423K** [1], the limit of the aluminium airframe. Early iterations used a range of partial differential equations to model the heat transfer. After performing accuracy and stability analysis of these methods, the Crank-Nicolson approximation was selected to calculate the above tile thickness, as it was most efficient.

8th April 2022

1.0 Introduction

Silica ceramic tiles were used on the space shuttle to form a crucial heat shield in attempt to disperse this heat. Sadly, in February 2003, the importance of this heat shield was demonstrated when damage to the tiles led to the destruction of the space shuttle Columbia upon re-entering the atmosphere [2]. The next generation of reusable spacecraft will use a similar layer of tiles for thermal protection. The tiles have excellent insulating properties however due to their area of application, are limited in thickness by weight restrictions. Therefore, being able to accurately model the heat transfer across these tiles during re-entry is extremely important.

2.0 Background

Most of the ceramic tiles on the space shuttle Columbia were made from a material called LI-900. This material was 99.9% pure silica glass fibres but was 94% air by volume. This meant that the tiles had very low thermal conductivity while being very lightweight. [3]

3.0 Method

To find a solution, a one-dimensional model of the tile was used, considering the temperature variations through the thickness of the tile. Initially, the forward differencing approximation to the heat equation was used to find a solution (**Equation 1**).

$$u_i^{n+1} = (1-2p)u_i^n + p(u_{i-1}^n + u_{i+1}^n) \tag{1}$$

$$p = \frac{\alpha \Delta t}{\Delta x^2} \tag{2}$$

Where: $\alpha = \frac{Therm \quad Conductivty}{Specific\ Heat\ Capacity \times Density}$

This equation was derived using a Taylor series approximation and can be referred to as an explicit method, as each new term for **u** can be calculated directly without using matrix algebra or iterating. The required boundary conditions were also set; the temperature at the outside of the tile, where $i = 1$, was defined using the available temperature data from NASA, shown in **Figure 1,** whereas the inside was initially considered as an insulated boundary.

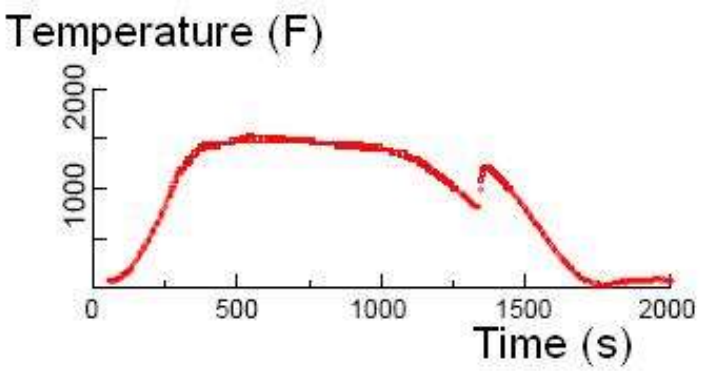

Figure 1 - *Measured temperature at tile #597 (outside temperature boundary)*

Dufort-Frankel is again explicit but is a multi-step method which relies on two approximations at each step; this results in a method which is inherently stable.

$$u_i^{n+1} = \frac{(1-2p)u_i^{n-1} + 2p(u_{i-1}^n + u_{i+1}^n)}{1+2p} \tag{3}$$

Dufort-Frankel Approximation

The next method, backward differencing, can be referred to as implicit, the benefits of which allow for the approximation to have better stability and accuracy. It approximates the heat equation by using values at the current timestep. This is achieved by looking back in time. Although it will produce simultaneous equations, they can be solved using a tridiagonal matrix.

$$u_i^{n+1} = \frac{u_i^n + p\left(u_{i-1}^{n+1} + u_{i+1}^{n+1}\right)}{1+2p} \tag{4}$$

Backward Differencing Approximation

Crank-Nicolson is an implicit method which averages the approximations of forward and backward differencing. However, the numerical heat equation approximation is taken at the midpoint between each timestep and averaging leading to improved accuracy. This is shown in **Equation 5** below.

$$u_i^{n+1} = \frac{(1-p)u_i^n + \frac{p}{2}\left(u_{i-1}^n + u_{i+1}^n + u_{i+1}^{n+1} + u_{i-1}^{n+1}\right)}{1-p} \tag{5}$$

Crank-Nicolson Approximation

Once all the methods were evaluated and implemented, a more realistic inside boundary condition for the tile was introduced. The Neumann boundary conditions uses the gradient of the heat transfer, normal to the inside wall of the tile. This means each method required a slightly different equation which are as followed.

$$u_1^{n+1} = (1-2p)u_1^n + 2p(u_2^n) \tag{6}$$

Forward Differencing Neumann Boundary

$$u_1^{n+1} = \frac{(1-2p)u_1^{n-1} + 4\,pu_2^n}{1+2p} \tag{7}$$

Dufort-Frankel Neumann Boundary

$$u_1^{n+1} = \frac{u_1^n + 2\,pu_2^{n+1}}{1+2p} \tag{8}$$

Backward Differencing Neumann Boundary at $i = 1$

$$u_{NX}^{n+1} = \frac{u_{NX}^{n} + 2\,pu_{NX-1}^{n+1}}{1+2p} \tag{9}$$

Backward Differencing Neumann Boundary at $i = NX$

$$u_{1}^{n+1} = \frac{(1-p)u_{1}^{n} + pu_{2}^{n} + pu_{2}^{n+1}}{1+p} \tag{10}$$

Crank Nicolson Neumann Boundary at $i = 1$

$$u_{NX}^{n+1} = \frac{(1-p)u_{NX}^{n} + pu_{NX-1}^{n} + pu_{NX-1}^{n+1}}{1+p} \tag{11}$$

Crank Nicolson Neumann Boundary at $i = NX$

4.0 Results and Discussion

Figure 2 and **Figure 3** below are used to conduct the stability and accuracy analysis of the four methods. **Figure 2** plots the inside boundary temperature against timestep size and, as expected from **Equation 2**, larger timesteps lead to instability within all methods. Therefore, reducing timestep increases the accuracy of the solution. However, this results in a significant increase in computational time and reduced the efficiency of the model.

Due to the high accuracy at very small timesteps, the point of convergence of the methods was selected as the true value of the simulation. To balance accuracy with its run time, a timestep was selected from the boundary where the outputs of all methods are within **1%** of the true value. This produced an ideal timestep of **8s** as seen in **Figure 2**.

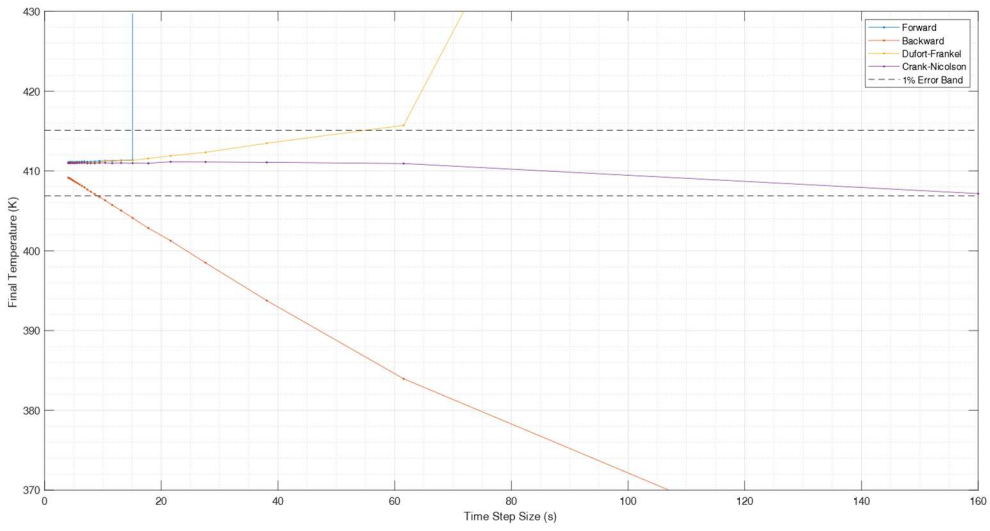

Figure 2 – *Inside Boundary Temperature against Timestep Size*

A similar approach was used when performing analysis on the effect of spatial step size on stability and accuracy. In this case, increasing size of spatial step has an inverse relationship with accuracy. As well as reducing the efficiency of the simulation, reducing spatial step also reduces the stability as supported by **Equation 2**. Again, error bars were plotted at **1%** from the true value to select the ideal spatial step size which, in **Figure 3**, is shown to be **0.016m**.

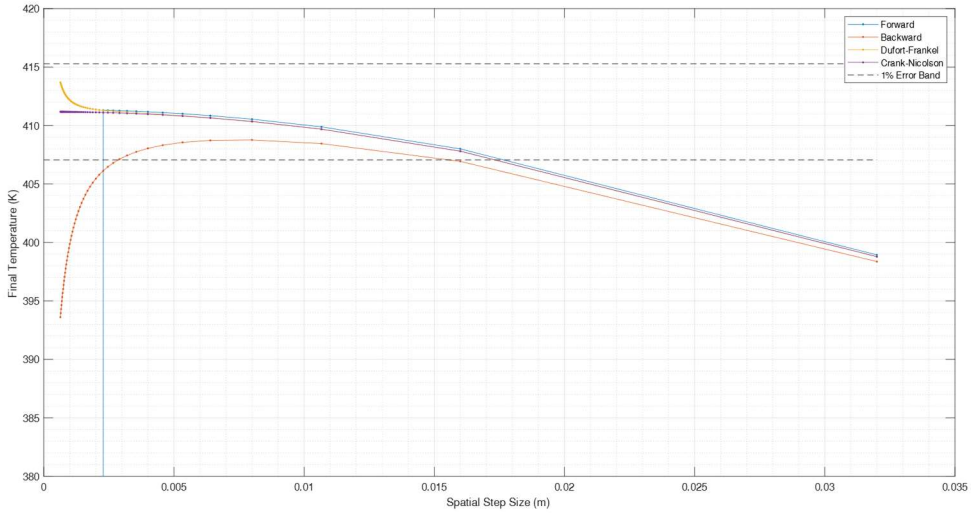

Figure 3 – *Inside Boundary Temperature against Spatial Step Size*

To optimise the model, the methods were compared so tile thickness could be calculated as accurately and efficiently as possible. When the error of each method was compared, the Big 'O' notation was used and in general, the accuracy increases with greater orders of error.

As can be seen above, all the methods have inaccuracies. This is because the derivations ignored the higher order terms within the Taylor series approximation. The Forward and Backward Differencing methods had first order accuracy in time and second order accuracy in space. Dufort-Frankel improves on these methods with second order time accuracy however has an additional induced error. Crank-Nicolson also has this improvement but avoids the induced error. Using Von Neumann stability analysis, it can be shown that when the forward method is unstable, **Equation 12** is true.

$$\Delta t > \frac{\Delta x^2}{2\alpha} \tag{12}$$

The forward differencing method was shown to be unsuitable due to this instability, which can be seen in **Figure 2** and **Figure 3**. This meant the method would only work for small time steps which caused it to be inefficient. Dufort-Frankel is again another explicit method and as shown in **Figure 2** it is unconditionally stable. However, the earlier mentioned induced error causes an oscillatory nature at larger timesteps. The two implicit methods are inherently stable, although the backwards differencing shares the same accuracy as the explicit forward method. The Crank-Nicolson method proved the most accurate at all timesteps and spatial steps at the slight expense of complexity,

however at larger timesteps could produce small oscillations. This method was consequently selected for modelling the required tile thickness.

Although a spatial step size of **0.016m** and timestep size of **8s** were selected as ideal values, solely using Crank-Nicolson would allow timestep size to be increased to **160s**. This would greatly increase simulation efficiency whilst maintaining a high level of accuracy.

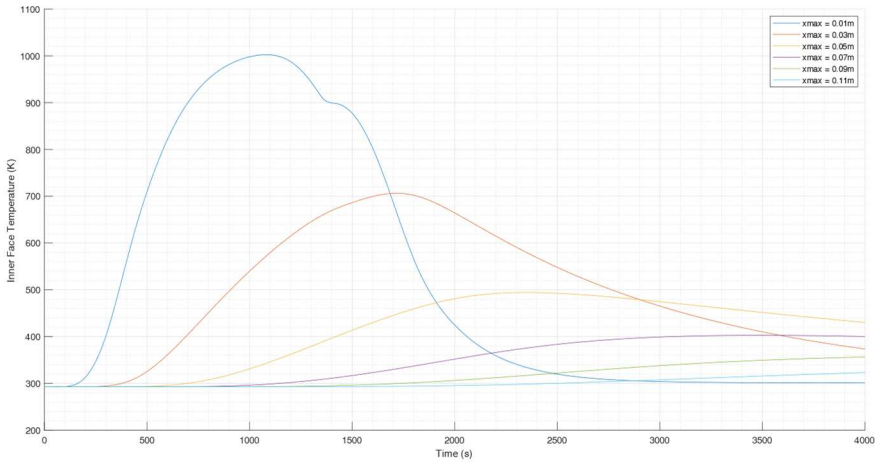

Figure 4 - *Inside Boundary Temperature for Range of Thicknesses*

Figure 4 is used to analyse the behaviour of the heat transfer through the extent of the tile as thickness changes. It can clearly be seen that for thinner tiles, the inside temperature profiles closely followed the output. The thin tiles heated up quickly but also dissipated the heat rapidly. The opposite was seen to be true for the thicker tiles. The major downside to a using a thick tile is the increase in weight and cost which is unacceptable to meet the requirements of the application.

Using Crank-Nicolson and simulation constants discussed, a minimum tile thickness of **64mm** was calculated. The result of this is shown below in **Figure 4**.

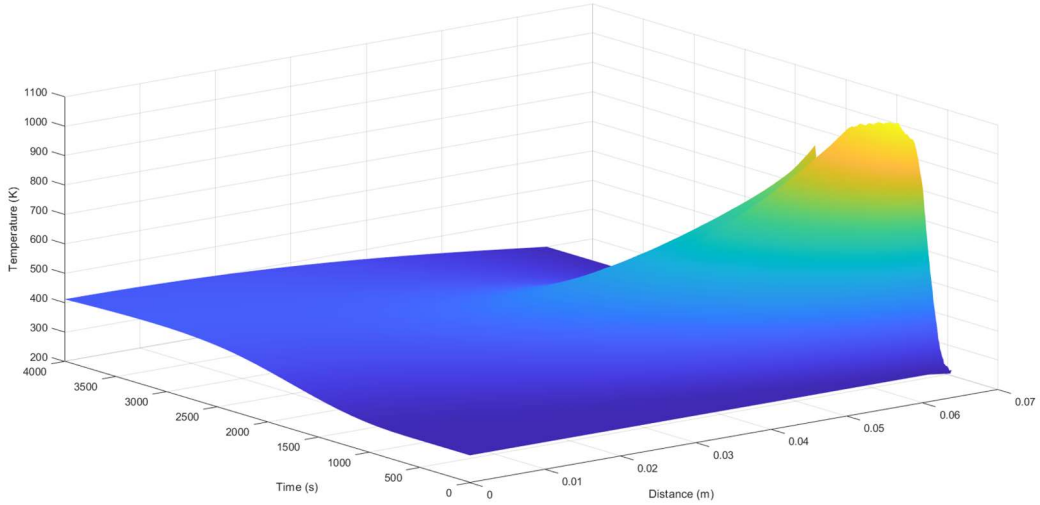

Figure 4 – *3D Temperature Profile*

Figure 5 shows the temperatures of the two boundary faces of the tile. It supports the calculated thickness of the tile as it maintains temperature below the threshold of the aluminium frame.

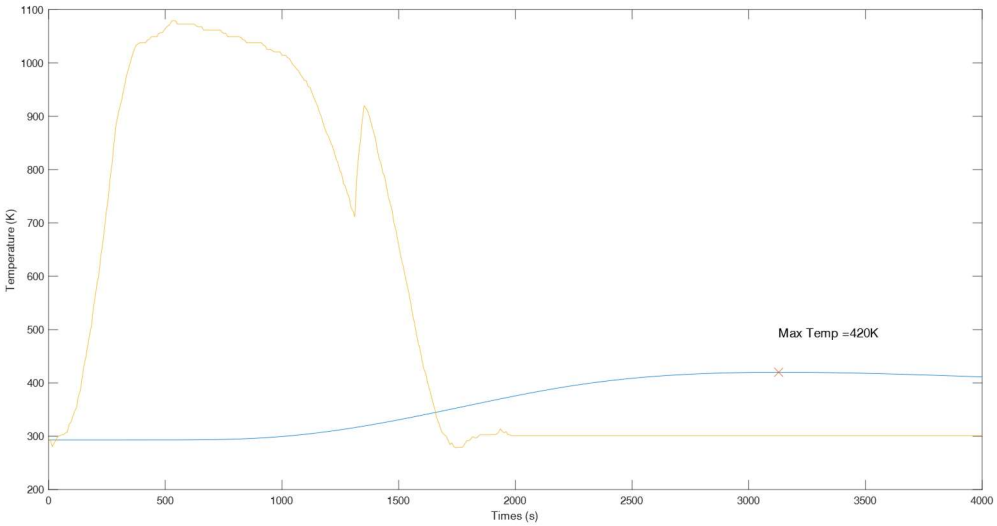

Figure 5 – *Temperature Profile for 0.05m Tile Thickness*

Although the model gives a good estimate, multiple external factors limit the final accuracy. First, it is assumed that the experienced temperature will be the same during each re-entry. However, due to the potential variations in entry speed, flight path and atmospheric conditions, this is unlikely true, and an error margin should be considered. Another assumption made is that all tiles are identical whereas in reality, during re-entry the tiles may degrade or be damaged by debris which would change the dimensions of the tiles and consequently effect the accuracy of the model. Finally, material properties are considered to be constant which is unlikely to be a fair assessment. The tiles are made in batches and are expected to have a long-life expectancy which could introduce variance immediately and in the longer term respectively.

5.0 Enhancements

Once the initial requirements where fulfilled, additional aspects were added to improve the flexibility and usability of the model. The initial iteration of the program required the user to modify the code to update the material properties of the tile. This was corrected by increasing the number of inputs to the shuttle.m function which allows the user to choose values quickly and simply for specific heat capacity, density, and thermal conductivity of the tile.

To improve the speed and accuracy of the programme, a function was designed to automatically read the temperature graphs produced by the NASA flight data. Before the implementation of this function, the user had to manually select data points from these graphs. This was inefficient as well as inaccurate due to human error and fewer data points being selected.

The functionality to automatically collect temperature data allowed for the possibility of multiple sensor readings being made available to the user. The NASA flight experiment had sensors positioned at multiple points across the spacecraft which is important because tiles did not receive

uniform heating across the airframe. Sensor selection was added as a further input to the shuttle.m function so the user can choose the most appropriate heating profile for tile modelling.

The shooting method function was implemented to calculate the optimum tile thickness for a desired maximum temperature of the inside face of the tile. The shooting method allows for BVP problems to be solved by iteratively reducing the error value of previous guesses. Initially two guesses for tile thickness are input, and each guess' resulting output is compared to the desired temperature. These errors are then scaled to predict a more accurate guess and the process is repeated. The following equations describe this method:

$$Z_1 = Initial\ guess\ 1 \qquad \varepsilon_1 = Z_1 - Z_D$$

$$Z_2 = Initial\ guess\ 2 \qquad \varepsilon_2 = Z_2 - Z_D$$

$$Z_{n+1} = Z_2 - \varepsilon_2 \left(\frac{Z_2 - Z_1}{\varepsilon_2 - \varepsilon_1} \right)$$

As can be seen in **Figure 7** below, the improvements made to increase the flexibility of the model caused the functions to have a great number of inputs. Despite this being a clear positive for one of the aims of the enhancements it is at a cost to the usability.

```
function [x, t, u, maxTemp, pos] = shuttle(tmax, nt, xmax, nx, method, thermCon, density, specHeat, sensor)

function [xmax, guess, result, idx, pos] = shootingxmax(intTemp,tmax, nt, nx, method, thermCon, density, specHeat, sensor)
```

Figure 7 – *Shuttle.m and shootingxmax.m Function Inputs*

Therefore, a MATLAB App was designed with the aim to greatly increase the ease of use of the model. Simulation conditions and tile material properties could be easily input into numerical boxes at the top of the app while method and sensor selection were made available through two drop down menus. Finally, the app was split into two tabs to allow the user to simply run the model using either the main tile simulation or the shooting method to calculate an optimal tile thickness. This app can be seen below in **Figure 8** and **Figure 9**.

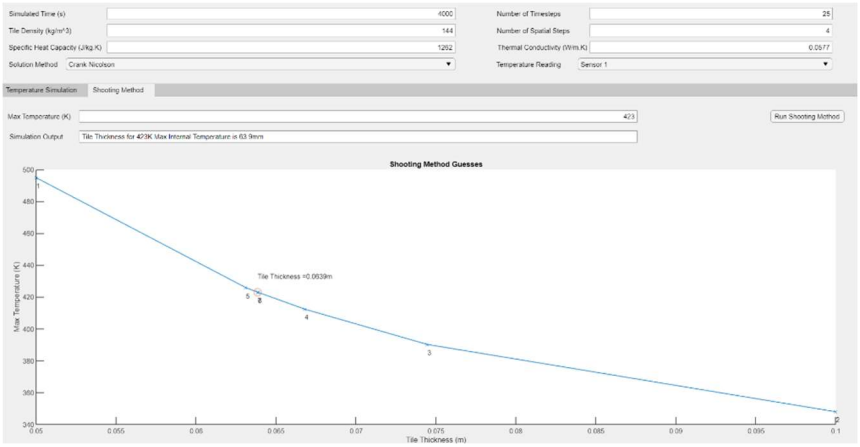

Figure 8 App Shooting Method

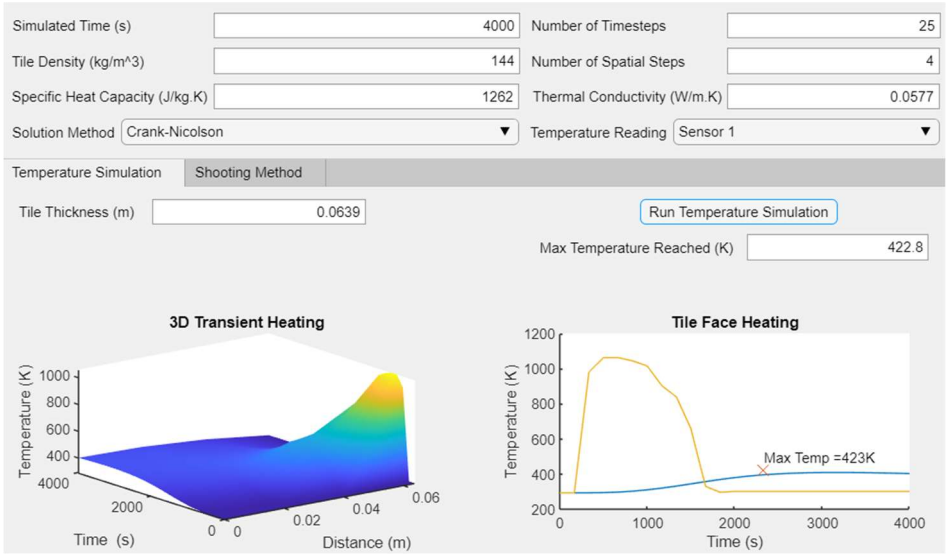

Figure 9 App Temperature Simulation

7.0 Conclusions

This report has discussed the process undertaken to develop a MATLAB model for heat experienced through a thermal insulation tile during re-entry. The Crank-Nicolson heat equation approximation was selected as the best modelling method due to its superior accuracy and efficiency. This resulted in a tile thickness of **0.064m** being calculated as optimal to protect the shuttle during re-entry. The flexibility and ease of use of the model was improved through the implementation of an App. In further iterations of the model, more detail regarding the material properties and structure could be considered to avoid some of the assumptions made during this design process.

8.0 References

1. Web.archive.org. 2022. *Shuttle Thermal Protection System (TPS)*. [online] Available at: <https://web.archive.org/web/20060826033923/http://www.centennialofflight.gov/essay/Evolution_of_Technology/TPS/Tech41.htm> [Accessed 6 April 2022].

2. Howell, E., 2022. Columbia Disaster: What Happened, What NASA Learned. [online] Space.com. Available at: https://www.space.com/19436-columbia-disaster.html [Accessed 6 April 2022].

3. Nasa.gov. 2022. [online] Available at: <https://www.nasa.gov/centers/kennedy/pdf/167473main_TPS-06rev.pdf> [Accessed 6 April 2022].

Appendix

Function – shuttle.m

```
function [x, t, u, maxTemp, pos] = shuttle(tmax, nt, xmax, nx, method, thermCon, density, specHeat, sensor)
% Function for modelling temperature in a space shuttle tile
%
% Input arguments:
% tmax      - maximum time (s)
% nt        - number of timesteps
% xmax      - total thickness (m)
% nx        - number of spatial steps
% method    - solution method ('Forward', 'Backward' etc)
% thermCon - material thermal conductivity (W/m.K)
% density   - material density (kg/m^3)
% specHeat - material specific heat capacity (J/kg.K)
% sensor    - select temperature sensor data to use
%
% Return arguments:
% x         - distance vector (m)
% t         - time vector (s)
% u         - temperature matrix (K)
% maxTemp   - maximum temperature reached on internal face of tile
% pos       - variable for position in time
%
% For example, to perform a  simulation with 501 time steps
%   [x, t, u, maxTemp, pos] = shuttle(4000, 501, 0.05, 21, 'Crank-Nicolson', 0.0577, 144, 1262, 'Sensor 1');
%

alpha = thermCon/(density * specHeat);

% Run autoplottemp.m
autoplottemp(sensor)
% Load temperature data from file
load temp.mat

% Initialise variables
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);
u([1 2], :) = 293;
i = 2:nx-1;
maxTemp = 0;

% Step through time
for n=2:nt-1

    % Use interpolation to get temperature at time vector t
    % and store it as boundary vectors.
    R = interp1(xScale, yScale, t, "linear", "extrap");

    % Select method and run simulation
```

```
    % Select method and run simulation
    switch method
        case 'Forward'
            u(n+1, nx) = R(n+1); % Outside boundary condition

            u(n+1,1) = (1-2*p) * u(n,1) + 2*p*u(n,2); % Neumann boundary condition
            u(n+1,i) = (1-2*p) * u(n,i) + p * (u(n,i-1) + u(n,i+1)); % Forward
differencing

        case 'Dufort-Frankel'
            u(n+1, nx) = R(n+1); % Outside boundary condition

            u(n+1,1) = ((1-2*p) * u(n-1,1) + 4*p*u(n,2)) / (1 + 2*p); % Neumann boundary
condition
            u(n+1,i) = ((1-2*p) * u(n-1,i) + 2*p*(u(n,i-1) + u(n,i+1))) / (1 + 2*p); %
Dufort-Frankel approximation

        case 'Backward'
            u(n+1, nx) = R(n+1); % Outside boundary condition

            % Calculate internal values using backward differencing
            b(1) = 1 + (2 * p);
            c(1) = -2 * p;
            d(1) = u(n,1); % Neumann boundary condition
            a(i) = -p;
            b(i) = 1 + 2*p;
            c(i) = -p;
            d(i) = u(n,2:nx-1); % Neumann boundary condition
            a(nx) = -2 * p;
            b(nx) = 1 + (2 * p);
            d(nx) = R(n+1);

            % Runs tdm.m
            u(n+1,:) = tdm(a,b,c,d);

        case 'Crank-Nicolson'

            % Calculate internal values using Crank-Nicolson
            b(1)    = 1 + p;
            c(1)    = -p;
            d(1)    = (1 - p) * u(n,1) + p * u(n,2); % Neumann boundary condition
            a(i) = -p/2;
            b(i) = 1 + p;
            c(i) = -p/2;
            d(i) = p/2*u(n,1:nx-2) + (1-p)*u(n,2:nx-1) + p/2*u(n,3:nx); % Neumann
boundary condition
            a(nx)   = 0;
            b(nx)   = 1;
            d(nx)   = R(n+1);
```

```
    switch method
        case 'Forward'
            u(n+1, nx) = R(n+1); % Outside boundary condition

            u(n+1,1) = (1-2*p) * u(n,1) + 2*p*u(n,2); % Neumann boundary condition
            u(n+1,i) = (1-2*p) * u(n,i) + p * (u(n,i-1) + u(n,i+1)); % Forward differencing

        case 'Dufort-Frankel'
            u(n+1, nx) = R(n+1); % Outside boundary condition

            u(n+1,1) = ((1-2*p) * u(n-1,1) + 4*p*u(n,2)) / (1 + 2*p); % Neumann boundary condition
            u(n+1,i) = ((1-2*p) * u(n-1,i) + 2*p*(u(n,i-1) + u(n,i+1))) / (1 + 2*p); % Dufort-Frankel approximation

        case 'Backward'

            % Calculate internal values using backward differencing
            b(1) = 1;
            c(1) = 0;
            d(1) = u(n,i(1)); % Neumann boundary condition
            a(i) = -p;
            b(i) = 1 + 2*p;
            c(i) = -p;
            d(i) = u(n,2:nx-1); % Neumann boundary condition
            a(nx) = 0;
            b(nx) = 1;
            d(nx) = R(n+1);

            % Runs tdm.m
            u(n+1,:) = tdm(a,b,c,d);

        case 'Crank-Nicolson'

            % Calculate internal values using Crank-Nicolson
            b(1)    = 1;
            c(1)    = 0;
            d(1)    = u(n,i(1)); % Neumann boundary condition
            a(i) = -p/2;
            b(i) = 1 + p;
            c(i) = -p/2;
            d(i) = p/2*u(n,1:nx-2)+ (1-p)*u(n,2:nx-1)+p/2*u(n,3:nx); % Neumann boundary condition
            a(nx)   = 0;
            b(nx)   = 1;
            d(nx)   = R(n+1);

            % Runs tdm.m
            u(n+1,:) = tdm(a,b,c,d);

        otherwise

        end

        % Updates value of maxTemp and its position in time
        if u(n,i(1)) > maxTemp
            maxTemp = u(n,i(1));
            pos = n*dt;
        end
    end

    end
```

Function – 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
```

Function – shootingxmax.m

```
function [xmax, guess, result, idx, pos] = shootingxmax(intTemp,tmax, nt, nx, ↙
method, thermCon, density, specHeat, sensor)
% Function for calculating required tile thickness for a maximum interal
% temperature during heating
%
% Input arguments:
% intTemp  - maximum temperature allowed for internal tile face (K)
% tmax     - maximum time (s)
% nt       - number of timesteps
% nx       - number of spatial steps
% method   - solution method ('Forward', 'Backward' etc)
% thermCon - material thermal conductivity (W/m.K)
% density  - material density (kg/m^3)
% specHeat - material specific heat capacity (J/kg.K)
% sensor   - select temperature sensor data to use
%
% Return arguments:
% xmax     - calculated tile thickness (m)
% guess    - array of guesses from shooting method
% result   - array of results from guesses
% idx      - index for guess and result arrays
% pos      - position in time of maximum temperature (s)

% Initialise variables
i = 0;
xmax = 0;
x1 = 0.05;
x2 = 0.1;
Error1 = 10;
Error2 = 10;
guess = [];
result = [];

% Calculates a tile thickness so maximum internal tile face temperature is not ↙
exceeded (accurate to within 1K)
while abs(Error1) && abs(Error2) > 1
    [~,~,~,MT1,~] = shuttle(tmax, nt, x1, nx, method, thermCon, density, specHeat, ↙
sensor);
    Error1 = MT1 - intTemp;
    % Runs shuttle for a secondary x guess and calculates Error2
    [~,~,~,MT2,~] = shuttle(tmax, nt, x2, nx, method, thermCon, density, specHeat, ↙
sensor);
    Error2 = MT2 - intTemp;
    % Calculates next guess, xMax, using previous xMax and errors
    guess = [guess xmax];
    xmax = x2 - (Error2*(x2 - x1)/(Error2-Error1));


    % Replaces the furthest previous guess with the new xmax
    if abs(Error1) < abs(Error2)
        x2 = xmax;
        result = [result MT1];
    else
```

```
        x1 = xmax;
        result = [result MT2];
    end

    % Puts the first two values in each array
    while i <1
        i = i + 1;
        guess = [0.05 0.1];
        result = [MT1 MT2];
    end
end

% Sorts guess into ascending order
[guess,idx] = sort(guess, 'ascend');
result = result(idx);

% Read pos from shuttle.m using calculated xmax
[~,~,~,~,pos] = shuttle(tmax, nt, xmax, nx, method, thermCon, density, specHeat, ↙
sensor);


% Output messages for improved usabilty
if pos == tmax
    fprintf('Maximum Temperature Occured at Final Timestep, Increase Simulated Time ↙
\n')
else
    fprintf('Tile Thickness for %dK Max Internal Temperature is %d mm.\n',intTemp,↙
round(xmax * 1e03))
end
```

Function – autoplottemp.m

```
function[] = autoplottemp(sensor)
% Function to automatically record temperature data from graphs
%
% Input arguments:
% sensor       - Temperature graph selection
%
% Image from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.1075
&rep=rep1&type=pdf


% Select sensor to run simulation with
switch sensor
    case 'Sensor 1'
        name = 'temp597';
        img=imread([name '.jpg']);

    case 'Sensor 2'
        name = 'temp468';
        img=imread([name '.jpg']);

    case 'Sensor 3'
        name = 'temp850';
        img=imread([name '.jpg']);

    case 'Sensor 4'
        name = 'temp730';
        img=imread([name '.jpg']);

    otherwise
        error(['Undefined Sensor: ' sensor])
end

% Read image to find pixels containing red and green
R = img(:,:,1);
G = img(:,:,2);

% Find black pixels
[y,x]=find(and(R<=0,G<=0));
% Find red pixels
[tempData,timeData]=find(and(R>=250,G<=180));


% Locate the graph axis from the image
count=1;
yaxis=[];
for k=1:length(x)-1

    if x(k)==x(k+1)
        count=count +1;
    end

    if x(k)~=x(k+1)
        count = 1;
```

```
    end

    if count>300
        yaxis=[count,x(k)];
        offset = mode(y);
    end
end

% Converting pixel data to time data in seconds
pixPerSec = (timeData(end)-yaxis(2))/2000;
xScale = (timeData - yaxis(2)) ./ pixPerSec;

% Converting pixel data to temperature data in kelvin
pixPerFar = (yaxis(end,1)/2000);
yScale = (((tempData - offset) ./ -pixPerFar) + 459.67) .*5/9;

% Average repeated values
[xScale,~,idx_2] = unique(xScale);
yScale = accumarray(idx_2,yScale,[],@mean);
yScale(end)=yScale(end-1);

% save data to .mat file with same name as image file
save('temp.mat','xScale','yScale')
```

Function – thickness.m

```
function[] = thickness()
% Function to plot temperature graphs for a range of tile thicknesses

% Initialise variables
nt = 501;
nx = 21;
tmax = 4000;
dt = tmax / (nt-1);
thermCon = 0.0577;
density = 144;
specHeat = 1262;
method = 'Crank-Nicolson';
sensor = 'Sensor 1';

% Vary tile thickness
for xmax = 0.01:0.02:0.11
    [~,~,u,~,~] = shuttle(tmax, nt, xmax, nx, method, thermCon, density, specHeat,↙
sensor);
    hold on
    plot(0:dt:tmax, u(:,1))
end
xlabel('Time (s)')
ylabel('Inner Face Temperature (K)')
legend('xmax = 0.01m','xmax = 0.03m','xmax = 0.05m','xmax = 0.07m','xmax = 0.09↙
m','xmax = 0.11m')
grid on
grid minor
hold off
```

Function – accuracy_dt.m

```
% Function to compare the accuracy of the four methods as dt changes
function[] = accuracy_dt

% Initialise variables
% xmax calculated using shooting method to limit maxTemp to 423K
xmax = 0.064;
nx = 21;
tmax = 4000;
i = 0;
thermCon = 0.0577;
density = 144;
specHeat = 1262;
sensor = 'Sensor 1';
x = [];

% Runs all four PDE methods through range of timestep sizes
for nt = 26:40:1001
    x = [x nt];
    i = i + 1;
    dt(i) = tmax/(nt-1);
    disp (['nt = ' num2str(nt) ', dt = ' num2str(dt(i)) ' s'])
    [~, ~, u] = shuttle(tmax, nt, xmax, nx, 'Forward', thermCon, density, specHeat,↙
sensor);
    uf(i) = u(end,1);
    [~, ~, u] = shuttle(tmax, nt, xmax, nx, 'Backward', thermCon, density,↙
specHeat, sensor);
    ub(i) = u(end,1);
    [~, ~, u] = shuttle(tmax, nt, xmax, nx, 'Dufort-Frankel', thermCon, density,↙
specHeat, sensor);
    ud(i) = u(end,1);
    [~, ~, u] = shuttle(tmax, nt, xmax, nx, 'Crank-Nicolson', thermCon, density,↙
specHeat, sensor);
    uc(i) = u(end,1);



end

% Plots the the inner surface temperature at 4000s against the timestep
plot(dt, [uf; ub; ud; uc],'.-')
hold on
plot([0 dt(1)], [0.99*uc(end) 0.99*uc(end)], '--',color=[0 0 0])
plot([0 dt(1)], [1.01*uc(end) 1.01*uc(end)], '--',color=[0 0 0])
hold off
xlabel('Time Step Size (s)')
ylabel('Final Temperature (K)')
ylim([370 430])
grid on
grid minor
legend ('Forward', 'Backward','Dufort-Frankel', 'Crank-Nicolson','1% Error Band')
```

Function – accuracy_nx.m

```
% Function to compare the accuracy of the four methods as dx changes
function[] = accuracy_nx

% Initialise variables
% xmax calculated using shooting method to limit maxTemp to 423K
xmax = 0.064;
nt = 501;
tmax = 4000;
i = 0;
thermCon = 0.0577;
density = 144;
specHeat = 1262;
sensor = 'Sensor 1';
x = [];

% Runs all four PDE methods through range of spacial step sizes
for nx = 3:2:101
    x = [x nx];
    i = i + 1;
    dx(i) = xmax/(nx-1);
    disp (['nx = ' num2str(nx) ', dx = ' num2str(dx(i)) ' m'])
    [~, ~, u] = shuttle(tmax, nt, xmax, nx, 'Forward', thermCon, density, specHeat,↙
sensor);
    uf(i) = u(end,1);
    [~, ~, u] = shuttle(tmax, nt, xmax, nx, 'Backward', thermCon, density,↙
specHeat, sensor);
    ub(i) = u(end,1);
    [~, ~, u] = shuttle(tmax, nt, xmax, nx, 'Dufort-Frankel', thermCon, density,↙
specHeat, sensor);
    ud(i) = u(end,1);
    [~, ~, u] = shuttle(tmax, nt, xmax, nx, 'Crank-Nicolson', thermCon, density,↙
specHeat, sensor);
    uc(i) = u(end,1);
end

% Plots the the inner surface temperature at 4000s against the timestep
figure(3)
plot(dx, [uf; ub; ud; uc],'.-')
hold on
plot([0 dx(1)], [0.99*uc(end) 0.99*uc(end)], '--',color=[0 0 0])
plot([0 dx(1)], [1.01*uc(end) 1.01*uc(end)], '--',color=[0 0 0])
hold off
ylim([380 420])
%xlim([0 0.01])
grid on
grid minor
xlabel('Spatial Step Size (m)')
ylabel('Final Temperature (K)')
legend ('Forward', 'Backward','Dufort-Frankel', 'Crank-Nicolson','1% Error Band')
```

App – Modeling_App.mlap

```
classdef Modeling_App_exported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        SimulatedTimesEditFieldLabel    matlab.ui.control.Label
        SimulatedTimesEditField         matlab.ui.control.NumericEditField
        TabGroup                        matlab.ui.container.TabGroup
        TemperatureSimulationTab        matlab.ui.container.Tab
        TileThicknessmEditFieldLabel    matlab.ui.control.Label
        TileThicknessmEditField         matlab.ui.control.NumericEditField
        RunTemperatureSimulationButton  matlab.ui.control.Button
        MaxTemperatureReachedKEditFieldLabel  matlab.ui.control.Label
        MaxTemperatureReachedKEditField  matlab.ui.control.NumericEditField
        UIAxes                          matlab.ui.control.UIAxes
        UIAxes2                         matlab.ui.control.UIAxes
        ShootingMethodTab               matlab.ui.container.Tab
        MaxTemperatureKEditField_2Label  matlab.ui.control.Label
        MaxTemperatureKEditField_2      matlab.ui.control.NumericEditField
        RunShootingMethodButton         matlab.ui.control.Button
        SimulationOutputEditFieldLabel  matlab.ui.control.Label
        SimulationOutputEditField       matlab.ui.control.EditField
        UIAxes3                         matlab.ui.control.UIAxes
        Label                           matlab.ui.control.Label
        NumberofTimestepsEditField      matlab.ui.control.NumericEditField
        NumberofSpatialStepsEditFieldLabel  matlab.ui.control.Label
        NumberofSpatialStepsEditField   matlab.ui.control.NumericEditField
        SolutionMethodDropDownLabel     matlab.ui.control.Label
        SolutionMethodDropDown          matlab.ui.control.DropDown
        ThermalConductivityWmKEditFieldLabel  matlab.ui.control.Label
        ThermalConductivityWmKEditField  matlab.ui.control.NumericEditField
        MaterialDensityLabel            matlab.ui.control.Label
        TileDensitykgm3EditField        matlab.ui.control.NumericEditField
        SpecificHeatCapacityJkgKEditFieldLabel  matlab.ui.control.Label
        SpecificHeatCapacityJkgKEditField  matlab.ui.control.NumericEditField
        TemperatureReadingDropDownLabel  matlab.ui.control.Label
        TemperatureReadingDropDown      matlab.ui.control.DropDown
    end


    properties (Access = public)
        x = []; % x coordinate vector
        t = []; % time vector
        u = []; % results matrix
        maxTemp = [];
        pos = [];


    end

    methods (Access = public)
```

```
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: RunTemperatureSimulationButton
        function RunTemperatureSimulationButtonPushed(app, event)
            % Initialise Variables From GUI
            tmax = app.SimulatedTimesEditField.Value;
            nt = app.NumberofTimestepsEditField.Value;
            xmax = app.TileThicknessmEditField.Value;
            nx = app.NumberofSpatialStepsEditField.Value;
            method = app.SolutionMethodDropDown.Value;
            thermCon = app.ThermalConductivityWmKEditField.Value;
            density = app.TileDensitykgm3EditField.Value;
            specHeat = app.SpecificHeatCapacityJkgKEditField.Value;
            sensor = app.TemperatureReadingDropDown.Value;
            dt = tmax / (nt-1);

            % Run shuttle.m
            [app.x, app.t, app.u, app.maxTemp, app.pos] = shuttle(tmax, nt, xmax, nx,↙
method, thermCon, density, specHeat, sensor);

            % Output to GUI
            app.MaxTemperatureReachedKEditField.Value = app.maxTemp;

            % Plot Results of Simulation
            surf(app.UIAxes, app.x, app.t , app.u)
            shading(app.UIAxes,"interp")

            plot(app.UIAxes2, 0:dt:tmax, app.u(:, 1));
            hold(app.UIAxes2, "on")
            text(app.UIAxes2, app.pos, (app.maxTemp + 75),['Max Temp =' num2str(round↙
(app.maxTemp)) 'K'], 'FontSize', 12)
            plot(app.UIAxes2, app.pos, app.maxTemp,'x','MarkerSize', 10)
            plot(app.UIAxes2, 0:dt:tmax, app.u(:, nx));
            hold(app.UIAxes2, "off")

        end

        % Button pushed function: RunShootingMethodButton
        function RunShootingMethodButtonPushed(app, event)
            % Initialise Variables From GUI
            tmax = app.SimulatedTimesEditField.Value;
            nt = app.NumberofTimestepsEditField.Value;
            nx = app.NumberofSpatialStepsEditField.Value;
            method = app.SolutionMethodDropDown.Value;
            thermCon = app.ThermalConductivityWmKEditField.Value;
```

```
            density = app.TileDensitykgm3EditField.Value;
            specHeat = app.SpecificHeatCapacityJkgKEditField.Value;
            sensor = app.TemperatureReadingDropDown.Value;
            intTemp = app.MaxTemperatureKEditField_2.Value;

            % Run shootingxmax.m
            [xmax, guess, result, idx, app.pos] = shootingxmax(intTemp,tmax, nt, nx,↙
method, thermCon, density, specHeat, sensor);

            % Plot Results of Simulation
            plot(app.UIAxes3,guess, result, '-x')
            hold(app.UIAxes3, "on")
            text(app.UIAxes3,xmax,(intTemp + 10), ['Tile Thickness =' num2str(round(xmax,↙
3,"significant")) 'm'], 'FontSize', 12)
            plot(app.UIAxes3,xmax,intTemp,'o','MarkerSize',10)
            for n = 1:max(idx)
                text(app.UIAxes3,guess(n), (result(n) - 5), num2str(idx(n)))
            end
            hold(app.UIAxes3, "off")

            % Output to GUI
            if app.pos == tmax
                app.SimulationOutputEditField.Value = ('Maximum Temperature Occured at↙
Final Timestep, Increase Simulated Time');
            else
                app.SimulationOutputEditField.Value = (['Tile Thickness for ' num2str↙
(intTemp) 'K Max Internal Temperature is ' num2str(round((xmax * 1e03),3,"significant"))↙
'mm']);
            end

        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'MATLAB App';

            % Create SimulatedTimesEditFieldLabel
            app.SimulatedTimesEditFieldLabel = uilabel(app.UIFigure);
            app.SimulatedTimesEditFieldLabel.HorizontalAlignment = 'right';
            app.SimulatedTimesEditFieldLabel.Position = [11 449 106 22];
            app.SimulatedTimesEditFieldLabel.Text = {'Simulated Time (s)'; ''};

            % Create SimulatedTimesEditField
```

```
            app.SimulatedTimesEditField = uieditfield(app.UIFigure, 'numeric');
            app.SimulatedTimesEditField.Position = [191 449 81 22];
            app.SimulatedTimesEditField.Value = 4000;

            % Create TabGroup
            app.TabGroup = uitabgroup(app.UIFigure);
            app.TabGroup.Position = [1 0 640 336];

            % Create TemperatureSimulationTab
            app.TemperatureSimulationTab = uitab(app.TabGroup);
            app.TemperatureSimulationTab.Title = 'Temperature Simulation';

            % Create TileThicknessmEditFieldLabel
            app.TileThicknessmEditFieldLabel = uilabel(app.TemperatureSimulationTab);
            app.TileThicknessmEditFieldLabel.HorizontalAlignment = 'right';
            app.TileThicknessmEditFieldLabel.Position = [11 280 103 22];
            app.TileThicknessmEditFieldLabel.Text = 'Tile Thickness (m)';

            % Create TileThicknessmEditField
            app.TileThicknessmEditField = uieditfield(app.TemperatureSimulationTab,↙
'numeric');
            app.TileThicknessmEditField.Position = [129 280 100 22];
            app.TileThicknessmEditField.Value = 0.05;

            % Create RunTemperatureSimulationButton
            app.RunTemperatureSimulationButton = uibutton(app.TemperatureSimulationTab,↙
'push');
            app.RunTemperatureSimulationButton.ButtonPushedFcn = createCallbackFcn(app,↙
@RunTemperatureSimulationButtonPushed, true);
            app.RunTemperatureSimulationButton.Position = [461 280 167 22];
            app.RunTemperatureSimulationButton.Text = 'Run Temperature Simulation';

            % Create MaxTemperatureReachedKEditFieldLabel
            app.MaxTemperatureReachedKEditFieldLabel = uilabel(app.↙
TemperatureSimulationTab);
            app.MaxTemperatureReachedKEditFieldLabel.HorizontalAlignment = 'right';
            app.MaxTemperatureReachedKEditFieldLabel.Position = [371 250 169 22];
            app.MaxTemperatureReachedKEditFieldLabel.Text = 'Max Temperature Reached↙
(K)';

            % Create MaxTemperatureReachedKEditField
            app.MaxTemperatureReachedKEditField = uieditfield(app.↙
TemperatureSimulationTab, 'numeric');
            app.MaxTemperatureReachedKEditField.Position = [551 250 74 22];

            % Create UIAxes
            app.UIAxes = uiaxes(app.TemperatureSimulationTab);
            title(app.UIAxes, '3D Transient Heating')
            xlabel(app.UIAxes, 'Distance (m)')
            ylabel(app.UIAxes, 'Time  (s)')
            zlabel(app.UIAxes, 'Temperature (K)')
```

```
            app.UIAxes.Position = [12 16 279 192];

            % Create UIAxes2
            app.UIAxes2 = uiaxes(app.TemperatureSimulationTab);
            title(app.UIAxes2, 'Tile Face Heating')
            xlabel(app.UIAxes2, 'Time (s)')
            ylabel(app.UIAxes2, 'Temperature (K)')
            zlabel(app.UIAxes2, 'Z')
            app.UIAxes2.Position = [344 16 278 192];

            % Create ShootingMethodTab
            app.ShootingMethodTab = uitab(app.TabGroup);
            app.ShootingMethodTab.Title = 'Shooting Method';

            % Create MaxTemperatureKEditField_2Label
            app.MaxTemperatureKEditField_2Label = uilabel(app.ShootingMethodTab);
            app.MaxTemperatureKEditField_2Label.HorizontalAlignment = 'right';
            app.MaxTemperatureKEditField_2Label.Position = [8 266 118 22];
            app.MaxTemperatureKEditField_2Label.Text = 'Max Temperature (K)';

            % Create MaxTemperatureKEditField_2
            app.MaxTemperatureKEditField_2 = uieditfield(app.ShootingMethodTab,↙
'numeric');
            app.MaxTemperatureKEditField_2.Position = [141 266 100 22];
            app.MaxTemperatureKEditField_2.Value = 423;

            % Create RunShootingMethodButton
            app.RunShootingMethodButton = uibutton(app.ShootingMethodTab, 'push');
            app.RunShootingMethodButton.ButtonPushedFcn = createCallbackFcn(app,↙
@RunShootingMethodButtonPushed, true);
            app.RunShootingMethodButton.Position = [478 266 132 22];
            app.RunShootingMethodButton.Text = 'Run Shooting Method';

            % Create SimulationOutputEditFieldLabel
            app.SimulationOutputEditFieldLabel = uilabel(app.ShootingMethodTab);
            app.SimulationOutputEditFieldLabel.HorizontalAlignment = 'right';
            app.SimulationOutputEditFieldLabel.Position = [11 230 101 22];
            app.SimulationOutputEditFieldLabel.Text = 'Simulation Output';

            % Create SimulationOutputEditField
            app.SimulationOutputEditField = uieditfield(app.ShootingMethodTab, 'text');
            app.SimulationOutputEditField.Position = [141 230 100 22];

            % Create UIAxes3
            app.UIAxes3 = uiaxes(app.ShootingMethodTab);
            title(app.UIAxes3, 'Shooting Method Guesses')
            xlabel(app.UIAxes3, 'Tile Thickness (m)')
            ylabel(app.UIAxes3, 'Max Temperature (K)')
            zlabel(app.UIAxes3, 'Z')
            app.UIAxes3.Position = [21 12 590 190];
```

```
            % Create Label
            app.Label = uilabel(app.UIFigure);
            app.Label.HorizontalAlignment = 'right';
            app.Label.Position = [341 449 119 22];
            app.Label.Text = 'Number of Timesteps';

            % Create NumberofTimestepsEditField
            app.NumberofTimestepsEditField = uieditfield(app.UIFigure, 'numeric');
            app.NumberofTimestepsEditField.Position = [511 449 80 22];
            app.NumberofTimestepsEditField.Value = 1001;

            % Create NumberofSpatialStepsEditFieldLabel
            app.NumberofSpatialStepsEditFieldLabel = uilabel(app.UIFigure);
            app.NumberofSpatialStepsEditFieldLabel.HorizontalAlignment = 'right';
            app.NumberofSpatialStepsEditFieldLabel.Position = [341 419 135 22];
            app.NumberofSpatialStepsEditFieldLabel.Text = 'Number of Spatial Steps';

            % Create NumberofSpatialStepsEditField
            app.NumberofSpatialStepsEditField = uieditfield(app.UIFigure, 'numeric');
            app.NumberofSpatialStepsEditField.Position = [511 419 79 22];
            app.NumberofSpatialStepsEditField.Value = 21;

            % Create SolutionMethodDropDownLabel
            app.SolutionMethodDropDownLabel = uilabel(app.UIFigure);
            app.SolutionMethodDropDownLabel.HorizontalAlignment = 'right';
            app.SolutionMethodDropDownLabel.Position = [11 359 92 22];
            app.SolutionMethodDropDownLabel.Text = 'Solution Method';

            % Create SolutionMethodDropDown
            app.SolutionMethodDropDown = uidropdown(app.UIFigure);
            app.SolutionMethodDropDown.Items = {'Forward', 'Backward', 'Crank-Nicolson',↙
'Dufort-Frankel'};
            app.SolutionMethodDropDown.Position = [118 359 153 22];
            app.SolutionMethodDropDown.Value = 'Forward';

            % Create ThermalConductivityWmKEditFieldLabel
            app.ThermalConductivityWmKEditFieldLabel = uilabel(app.UIFigure);
            app.ThermalConductivityWmKEditFieldLabel.HorizontalAlignment = 'right';
            app.ThermalConductivityWmKEditFieldLabel.Position = [341 389 166 22];
            app.ThermalConductivityWmKEditFieldLabel.Text = 'Thermal Conductivity (W/m.↙
K)';

            % Create ThermalConductivityWmKEditField
            app.ThermalConductivityWmKEditField = uieditfield(app.UIFigure, 'numeric');
            app.ThermalConductivityWmKEditField.Position = [511 389 79 22];
            app.ThermalConductivityWmKEditField.Value = 0.0577;

            % Create MaterialDensityLabel
            app.MaterialDensityLabel = uilabel(app.UIFigure);
            app.MaterialDensityLabel.HorizontalAlignment = 'right';
            app.MaterialDensityLabel.Position = [11 419 117 22];
```

```
            app.MaterialDensityLabel.Text = 'Tile Density (kg/m^3)';

            % Create TileDensitykgm3EditField
            app.TileDensitykgm3EditField = uieditfield(app.UIFigure, 'numeric');
            app.TileDensitykgm3EditField.Position = [191 419 81 22];
            app.TileDensitykgm3EditField.Value = 144;

            % Create SpecificHeatCapacityJkgKEditFieldLabel
            app.SpecificHeatCapacityJkgKEditFieldLabel = uilabel(app.UIFigure);
            app.SpecificHeatCapacityJkgKEditFieldLabel.HorizontalAlignment = 'right';
            app.SpecificHeatCapacityJkgKEditFieldLabel.Position = [11 389 171 22];
            app.SpecificHeatCapacityJkgKEditFieldLabel.Text = 'Specific Heat Capacity↙
(J/kg.K)';

            % Create SpecificHeatCapacityJkgKEditField
            app.SpecificHeatCapacityJkgKEditField = uieditfield(app.UIFigure, 'numeric');
            app.SpecificHeatCapacityJkgKEditField.Position = [191 389 80 22];
            app.SpecificHeatCapacityJkgKEditField.Value = 1262;

            % Create TemperatureReadingDropDownLabel
            app.TemperatureReadingDropDownLabel = uilabel(app.UIFigure);
            app.TemperatureReadingDropDownLabel.HorizontalAlignment = 'right';
            app.TemperatureReadingDropDownLabel.Position = [339 359 121 22];
            app.TemperatureReadingDropDownLabel.Text = 'Temperature Reading';

            % Create TemperatureReadingDropDown
            app.TemperatureReadingDropDown = uidropdown(app.UIFigure);
            app.TemperatureReadingDropDown.Items = {'Sensor 1', 'Sensor 2', 'Sensor 3',↙
'Sensor 4'};
            app.TemperatureReadingDropDown.Position = [490 359 100 22];
            app.TemperatureReadingDropDown.Value = 'Sensor 1';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = Modeling_App_exported

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
```

```
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```