MODELLING OF A SPACECRAFT HEAT SHIELD TILE

Mr D Jhugroo

12th April 2019

Abstract

The heat shield tile of a spacecraft was modelled in MATLAB, to predict the temperature though the thickness of a tile during the re-entry phase, so as to consequently choose a suitable tile thickness. This was done by automatically scanning temperature graphs corresponding to different tile locations on the spacecraft, and by computing the time and temperature data into a one-dimensional heat equation. Investigating the accuracy and stability of four different numerical methods: Forward differencing; Backward differencing; DuFort-Frankel; and Crank-Nicolson, the Crank-Nicolson method was found to be the most accurate and stable one. Suitable numbers of timesteps and of step sizes were identified and were used with the selected numerical method to calculate the optimum thickness of the different tiles. The maximum optimum thickness was found to be 6.412 cm for the tile at location 850. The effects of the assumptions made in the model were investigated, and improvements were suggested, together with the need to validate the numerical result.

1. Introduction [1][2][3][4]

Space shuttle launched by NASA, carried up to seven astronauts to space and back on Earth. Re-entering the Earth' atmosphere at more than 17,000 mph, atmospheric friction heated the external surface of the shuttle to temperatures as high as 3,000° F. Special thermal shields are required to protect the shuttle and its occupants, as the airframe, which is mainly made from aluminium, can only withstand a temperature of 350° F. High-temperature Reusable Surface Insulation (HRSI) tiles are used as thermal shields, as they are optimised for maximum emissivity, thereby improving heat rejection during the hot phase of re-entry.

Following the Columbia space shuttle disaster on February 1st, 2003, the accident was caused by the damage of the tiles on the left wing of the shuttle. This highlighted the necessity to model the variation of temperature along the thickness of the tile, during the design phase. To predict the temperature through the thickness of a tile during the re-entry, and to subsequently calculate a suitable tile thickness based on the temperature at the inner surface, a program was made in MATLAB. This involved the use of a partial differential equation (PDE) – The Heat Equation – which was solved using four different numerical methods; the accuracy and stability of which were investigated.

2. Theory [5]

2.1. The Heat Equation

The *Heat Equation*, shown in (1), is a partial differential equation (PDE), which defines temperature as a function of time and space in a solid.

$$\dot{u} = \alpha \nabla^2 u \tag{1}$$

The heat equation can be used in 1D, 2D and 3D, where D is a dimension in space. As in this case the heat equation would only be used in 1D (i.e. to represent temperature through the thickness of a uniform thick tile), the heat equation in 1D is shown in (2).

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \tag{2}$$

For a solid, the thermal diffusivity, α, is obtained from the thermal conductivity, k, the density, ρ, and the specific heat, C_p. This relationship is shown in (3).

$$\alpha = \frac{k}{\rho C_p} \tag{3}$$

2.2. The Forward Differencing Method

The forward differencing approximation to the 1D heat equation is given by

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} = \alpha \frac{u_{i-1}^n-2u_i^n+u_{i+1}^n}{\Delta x^2} \tag{4}$$

where *n* is the point in time, *i* is the spatial point, Δt is the timestep and Δx is the step size.

Rearranging (4), the value of u at the next timestep, for a spatial point, can be calculated from the current values of u at the local and adjacent spatial points.

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

The value of p is given by

$$p = \alpha \frac{\Delta t}{\Delta x^2} \tag{6}$$

Equation (5) is an explicit numerical method, known as the Forward Differencing Method.

The Forward Differencing Method has first order accuracy in time, and second order accuracy in space, i.e. $O(\Delta t, \Delta x^2)$. It also has limited stability as the solution goes unstable and oscillates with increasing timestep. For stability, p must be in the range, $0 < p < 0.5$.

2.3. The DuFort-Frankel Method

The central differencing approximation to the 1D heat equation is given by

$$\frac{u_i^{n+1}-u_i^{n-1}}{2\Delta t} = \alpha \frac{u_{i-1}^n - (u_i^{n-1}+u_i^{n+1})+u_{i+1}^n}{\Delta x^2} \tag{7}$$

Rearranging (7), the DuFort-Frankel method is given by

$$u_i^{n+1} = \frac{(1-2p)u_i^{n-1} + 2p(u_{i-1}^n+u_{i+1}^n)}{(1+2p)} \tag{8}$$

The DuFort-Frankel method is an explicit numerical method with an accuracy given by $O\left(\Delta t^2, \Delta x^2, \left(\frac{\Delta t}{\Delta x}\right)^2\right)$. This method is unconditionally stable but suffers from oscillatory behaviour at large time steps. However, as it is a multi-step method, initial conditions must be set for two timesteps.

2.4. The Backward Differencing Method

The backward differencing approximation to the 1D heat equation is given by

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} = \alpha \frac{u_{i-1}^{n+1}-2u_i^{n+1}+u_{i+1}^{n+1}}{\Delta x^2} \tag{9}$$

Rearranging (9), the Backward Differencing Method is given by

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

With three unknown values of u, a set of N simultaneous equations is obtained, where N is the number of spatial points. This can be expressed in matrix-vector form and solved using tri-diagonal matrix method to calculate u_i^{n+1}.

The Backward Differencing Method is an implicit numerical method, which has first order accuracy in time, and second order accuracy in space, i.e. $O(\Delta t, \Delta x^2)$. However, it is always stable, giving smooth results.

2.5. The Crank-Nicolson Method

The approximation to the 1D heat equation is given by

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} = \frac{\alpha}{2}\left(\frac{u_{i-1}^n-2u_i^n+u_{i+1}^n}{\Delta x^2} + \frac{u_{i-1}^{n+1}-2u_i^{n+1}+u_{i+1}^{n+1}}{\Delta x^2}\right) \tag{11}$$

Rearranging (11), the Crank-Nicolson method is given by

$$\left(-\frac{p}{2}\right)u_{i-1}^{n+1}+(1+p)u_i^{n+1}+\left(-\frac{p}{2}\right)u_{i+1}^{n+1} = \left(\frac{p}{2}\right)u_{i-1}^n+(1-p)u_i^n+\left(\frac{p}{2}\right)u_{i+1}^n \tag{12}$$

Like for the case of the Backward Differencing Method, the tri-diagonal matrix method is used to solve for u_i^{n+1}.

The Crank-Nicolson Method is an implicit numerical method, having second order accuracy in both time and space, i.e. $O(\Delta t^2, \Delta x^2)$. However, it can produce spurious, unphysical oscillations for large timesteps.

3. Program Structure

3.1. Generation of Temperature Profile

A function was made to calculate the temperature along the thickness of a tile, against time. The temperature was calculated at discrete points and its variation with time and thickness was plotted. The function ***shuttle.m*** performed this task, taking as inputs: the location number of the tile, the maximum time, the number of timesteps, the thickness of the tile, the number of spatial steps, the numerical method to compute the solution, and a Boolean to plot the variation of temperature. The output consisted of the time vector, the distance vector, and the calculated temperature matrix. To insulate the inner surface so that there was no heat flow through it, Neumann boundary condition was used. On the other hand, a Dirichlet boundary condition was used at the outer surface, where the temperature, which varied with time, was obtained from the linear extrapolation of a *timedata* vector and its corresponding *tempdata* vector. These vectors were loaded from a *.mat* file corresponding to the location number of the tile. Ultimately, the temperature profile was generated using the numerical method (i.e. Forward differencing, DuFort-Frankel, Backward differencing or Crank-Nicolson) stated in the input. The implicit numerical methods used the ***tdm.m*** function to solve the tri-diagonal matrices. The MATLAB code for the ***shuttle.m*** and the ***tdm.m*** functions can be found in *Appendix A* and *Appendix B* respectively.

3.2. Accuracy and Stability Analysis

A script was written to plot the inner surface temperature variation with timestep, at time being equal to 4000s, using the four different numerical methods. The ***MethodCompare_nt.m*** script performed this task by running the ***shuttle.m*** function using the four different methods for each value of *nt* (number of timesteps), where *nt* was varied from 41 to 1001 in steps of 20. This was done using *nx* (number of spatial steps) being equal to 21, the thickness of the tile being 0.05m, and the maximum running time being 4000s. The MATLAB code for the ***MethodCompare_nt.m*** script can be found in *Appendix C.*

Another similar script was written to plot the inner surface temperature variation with step size, again at time equal to 4000s and using the four different numerical methods. This was done by the ***MethodCompare_nx.m*** script, but this time varying *nx* from 5 to 51 in unity steps and using *nt* being fixed at 501. The MATLAB code for the ***MethodCompare_nx.m*** script can be found in *Appendix D.*

3.3. Variation of Tile Thickness

A function was made to plot the inner surface temperature variation with thickness of the tile, against time; and to find the minimum thickness of the tile so that the inner surface temperature does not exceed 450K (350°F). The function ***inSurfTemp.m*** performed this task, taking as inputs: the location number of the tile, the maximum time, the number of timesteps, the number of spatial steps, and the numerical method to compute the solution. Varying the thickness of the tile from 0.05m to 0.07m in steps of 0.001m, these parameters were used in the ***shuttle.m*** function to obtain the corresponding time and temperature values. Comparing the inner surface temperature at all times with 450K, for each value of thicknesses, the optimum thickness value was calculated and provided as the output of the function, while also being displayed in the command window. The MATLAB code for the ***inSurfTemp.m*** function can be found in *Appendix E*.

4. Enhancements and Extensions

4.1. Automatic Scan

A function was made to automatically read and save the time and temperature data from image graphs, into a *.mat* file. The function ***plottemp.m*** performed this task, taking the location number of the tile as input. The image corresponding to the location number of the tile, was then selected and the pixel coordinates corresponding to the colour red and black were detected through their RGB values. As such, the temperature plot, the axes and consecutively the origin, were identified. Using the fact that the end limit of each axis corresponded to the value being 2000, the pixel coordinates of the temperature curve, were converted into time/temperature coordinates. Hence the time data and the temperature data were obtained. Converting the temperature data from Fahrenheit to Kelvin, the *timedata* and *tempdata* were saved into a *.mat* file with the name of the location number of the tile investigated. The MATLAB code for the ***plottemp.m*** function can be found in *Appendix F*.

4.2. Tile Location Selection

The graphs for the temperature variation during re-entry phase, at different tile locations, were included in the program. The different MATLAB functions and scripts were modified to take the location number of the tile as an additional input. This enabled the temperature profile at different locations on the space shuttle, to be investigated and the respective optimum thickness values were calculated, thereby enhancing the modelling accuracy. The different graphs corresponding to different tile locations on the spacecraft, are shown in *Figure 1*. [6]

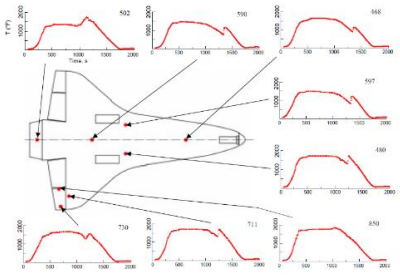


Figure 1 *- STS-96 windward thermocouple data from entry interface (400,000 ft) to landing*

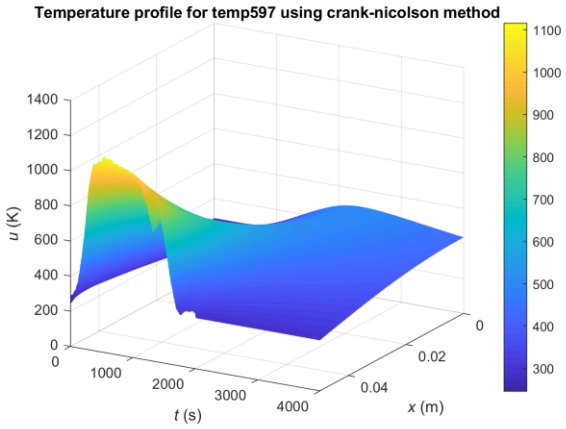
4.3. Optimum Tile Thickness

The ***shootingMethod.m*** function was made to calculate the minimum thickness required for the inner surface temperature, at all times, not to exceed 450K. This function uses a tactical approach to trial and error by making intelligent guesses. To improve the accuracy of the result, *nt* and *nx* were set to be 801 and 34 respectively, and the Crank-Nicolson method was used, in the incorporated ***shuttle.m*** function. The MATLAB code for the ***shootingMethod.m*** function can be found in *Appendix G*.

Furthermore, a script, ***tilesThick.m*** was written to calculate the optimum thickness for each of the different tile locations. The MATLAB code for the ***tilesThick.m*** script can be found in *Appendix H*.

5. Results and Discussion

5.1. Temperature Profile



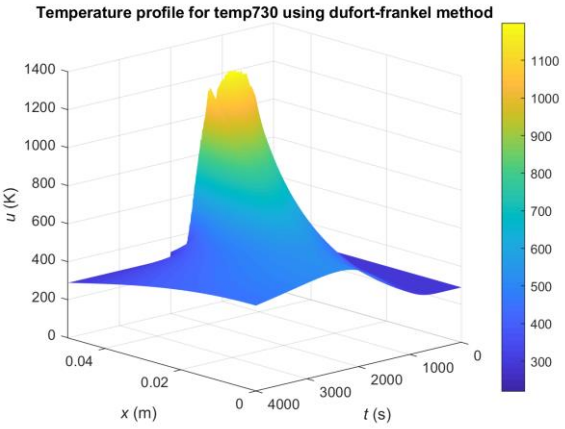
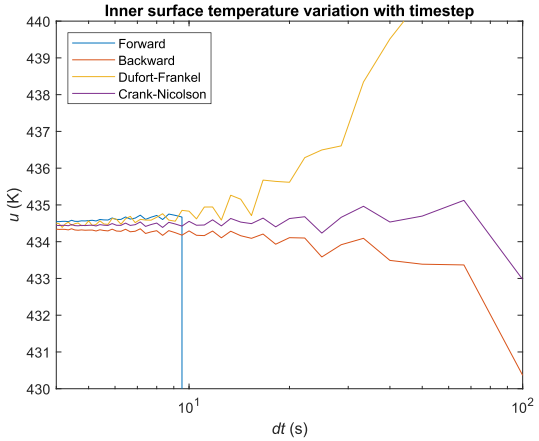


Figure 2 *- Temperature profile for tile 597 - Crank-Nicolson*

Figure 3 *- Temperature profile for tile 730 - DuFort-Frankel*

The temperature variation with time and distance through the tile was plotted for tiles 597 and 730 in *Figure 2* and *Figure 3* respectively. An implicit method, the Crank-Nicolson method, was used for the first one, and an explicit method, the DuFort-Frankel method, was used for the second one. As shown in *Figure 2*, the outer surface of the tile has temperature values corresponding to the scanned temperature graph for tile 597, for time up to 2000s. This indicated correct implementation of the Dirichlet boundary condition. Moving towards the inner surface of the tile, as shown in *Figure 3*, the temperature for time less than 2000s, was observed to decrease rapidly, while the temperature for time greater than 2000s, was seen to increase gradually. This showed the conduction of heat within the tile to maintain uniform temperature along its thickness.

5.2. Accuracy and Stability Analysis



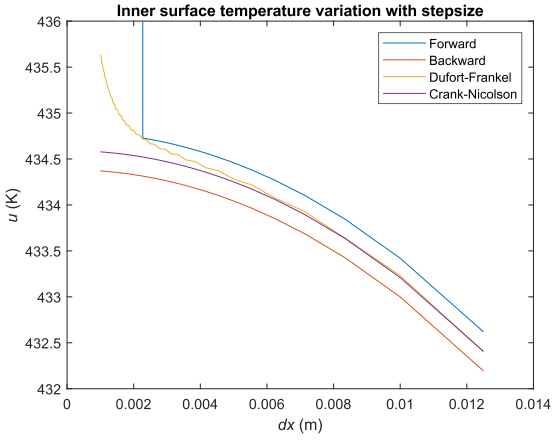


Figure 4 *- Inner surface temperature variation with timestep*

Figure 5 *- Inner surface temperature variation with step size*

The inner surface temperature variation, with timestep and with step size were plotted in Figure 4 and Figure 5 respectively, for time equal to 4000s, using the four different numerical methods.

Using the forward differencing method, for the variation in timestep, the timestep at which the method goes unstable, was calculated using equation (6), where the maximum value of *p* for stability is 0.5. As such, the solution was calculated to go unstable at a timestep of 9.79s. This was confirmed by the plot in *Figure 4*, where the solution of the forward differencing method went unstable at this value of timestep. For the variation of step size, the minimum value of step size for stability was calculated to be 0.00226. This was also confirmed by the plot in *Figure 5*. Hence, the forward differencing method was shown to have limited stability. Moreover, the plot for the forward method in *Figure 4* was observed to deviate the most from the horizontal. This deviation proved the forward differencing method to have limited accuracy, as per its first order accuracy in time and second order accuracy in space.

Also having first order accuracy in time and second order accuracy in space, the backward differencing method showed similar limited accuracy in *Figure 4*. However, the method was still stable at large timesteps and step sizes, always giving smooth results. This confirmed the stability prediction.

With an accuracy given by $O\left(\Delta t^2, \Delta x^2, \left(\frac{\Delta t}{\Delta x}\right)^2\right)$, the DuFort-Frankel method was found to have a very good accuracy. However, as shown in *Figure 5*, as the step size was decreased, the solution was found to increasingly lose accuracy. This can be explained by the $\left(\frac{\Delta t}{\Delta x}\right)^2$ term in the order of accuracy. Moreover, as shown in *Figure 4*, and as predicted by theory, the method was unconditionally stable, but suffered from oscillatory behaviour at large timesteps.

The Crank-Nicolson method was found to be the most accurate numerical method, supporting the fact that it has second order accuracy in both time and space. Furthermore, as shown in *Figure 4*, it featured unconditional stability, with only some small and unphysical oscillations for large timesteps.

As a result, the Crank-Nicolson method was chosen to perform the modelling and suitable values for *nt* and *nx* were calculated by analysing the plots in *Figure 4* and *Figure 5* to find the timestep and step size at which, least deviation from the solution was obtained. With Δt = 5 and Δx = 0.0015, *nt* and *nx* were calculated to be 801 and 34 respectively.

5.3. Assumptions

5.3.1. The properties of the material in the tile were assumed to be uniform and isotropic. However, variation of density, thermal conductivity and specific heat, along the thickness of the tile, would result in a different temperature variation along its thickness. Samples at different locations along thickness could be taken and their different properties calculated. The spatial changes in material properties could then be modelled at each node of transition.

5.3.2. The modelling was performed as a one-dimensional problem, as only heat flow across the thickness of the tile was considered. As such, the tile was assumed to behave like an infinite plate of finite thickness. The heat equation could be used in three-dimension, modelling the tile as a solid block, where heat is conducted across its length and width, to and from adjacent tiles.

5.3.3. The temperature at the outer surface of the tile was assumed to reach the outside temperature instantaneously. However, the surface temperature would practically increase more gradually from its initial value. The effect of heat transfer by radiation at the surface could be modelled as radiation is the dominant factor of heat transfer.

5.3.4. The outer surface temperature was assumed to be ambient for time greater than 2000s. However, in practice, the temperature decreased gradually instead of the abrupt drop in value. The modelling could be improved by recording the temperature variation for a longer time interval, until ambient temperature was reached.

5.4. Validation

Despite the expected physical behaviour of the model, the results should be compared against some experimental values. As such, a very large piece of tile having an equal thickness, could be heated at on side using a heater producing temperatures equivalent to the re-entry phase temperature variation. By measuring the temperature at the other side of the tile, where the tile is properly insulated, the comparison could be made, and the model results validated accordingly.

5.5. Suitable Tile Thickness

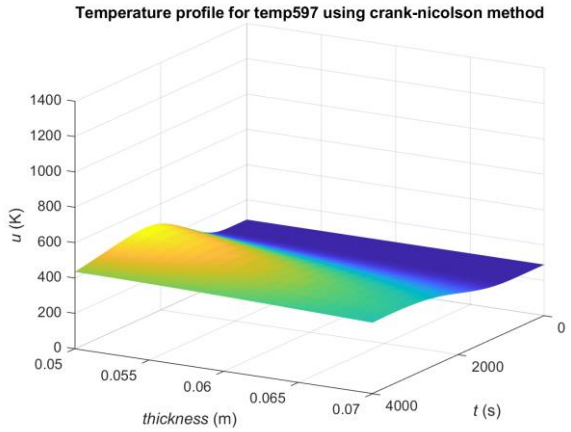


***Figure 6** - Inner surface temperature variation with thickness*

As demonstrated by *Figure 6*, the variation of temperature with time, along the thickness of the tile, decreased in intensity, with increasing thickness. However, as the airframe of the space shuttle can only withstand a maximum temperature of 450K (350°F), the minimum thickness of the tile to be used, must be found. This optimum thickness was found to be 0.059m, for the 597 tile, by varying the thickness of the tile in steps of 0.001, until a temperature of 450K was reached. To further increase the accuracy and precision of this thickness value, and to reduce computation time, a shooting method function was used in MATLAB and the optimum thickness values were calculated for each tile location. The results are shown in *Figure 7*.

```
Command Window
  >> tilesThick
  Thickness of tile required for location 468 is 0.059634 m
  Thickness of tile required for location 480 is 0.063123 m
  Thickness of tile required for location 502 is 0.056855 m
  Thickness of tile required for location 590 is 0.055775 m
  Thickness of tile required for location 597 is 0.05801 m
  Thickness of tile required for location 711 is 0.06303 m
  Thickness of tile required for location 730 is 0.06119 m
  Thickness of tile required for location 850 is 0.064121 m
fx >>
```

Figure 7 - Optimum thickness for different tile locations

6. Conclusion

On comparing the different numerical methods to model the heat shield, the forward differencing method was found to have limited accuracy and stability. Despite the good level of accuracy of the DuFort-Frankel method, its accuracy was found to decline with increasing timestep. While the backward differencing method always generated smooth results, its limited accuracy in time, led to the Crank-Nicolson method being the most accurate and stable numerical method. As such, the Crank-Nicolson method, together with suitable parameters, nt = 801 and nx = 34, – found by analysing the method plot with change in timestep and step sizes – were used to calculate the optimum thickness of the different tiles of the space shuttle. The maximum optimum thickness was found to be 6.412 cm for the tile at location 850. Assumptions made in the modelling, exposed the inaccuracies of the model: the

problem was modelled in only one-dimension; the variation of temperature was very abrupt at times, ignoring the latency in practice; the material properties were considered to be uniform and isotropic. Improvements were suggested, and the need to validate the result was investigated.

7. References

[1] NASA. (2019). What Was the Space Shuttle?. [online] Available at: https://www.nasa.gov/audience/forstudents/5-8/features/nasa-knows/what-is-the-space-shuttle-58.html [Accessed 5 Apr. 2019].

[2] Nasa.gov. (2019). Orbiter Thermal Protection System. [online] Available at: https://www.nasa.gov/centers/kennedy/pdf/167473main_TPS-06rev.pdf?fbclid=IwAR1rWuVYMop2T7nsFf_bee0e9XR3Gzir-e-WReKGENjfas3tjlCOt99DUj0 [Accessed 5 Apr. 2019].

[3] Asminternational.org. (2019). NASA's SHUTTLE PROGRAM. [online] Available at: https://www.asminternational.org/documents/10192/1880399/amp16101p069.pdf/ee0857f8-57a3-40b6-81eb-01efc9023025 [Accessed 5 Apr. 2019].

[4] Howell, E. (2019). Columbia Disaster: What Happened, What NASA Learned. [online] Space.com. Available at: https://www.space.com/19436-columbia-disaster.html [Accessed 5 Apr. 2019].

[5] Johnston, D N. (2019). Modelling Techniques II, Engineering Applications and Numerical Solution of Partial Differential Equations. [Accessed 5 Apr. 2019].

[6] Blanchard, R & Wilmoth, Richard & E. Glass, Christopher & Ronald Merski, N & A. Berry, Scott & J. Bozung, Timothy & Tietjen, Alan & Wendt, Jodean & Dawson, Don. (2001). Infrared Sensing Aeroheating Flight Experiment: STS-96 Flight Results. Journal of Spacecraft and Rockets. 38. 10.2514/2.3713. [Accessed 9 Apr. 2019].

Appendices

Appendix A: MATLAB code for *shuttle.m*

```
function [x, t, u] = shuttle(name,tmax, nt, xmax, nx, method, doplot)
% Function for modelling temperature in a space shuttle tile
% D N Johnston  30/1/19
% Modified by D Jhugroo 10/3/19
%
% Input arguments:
% name   - name of image to scan
% 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

% 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.

name = num2str(name);
name = ['temp' name];
plottemp(name);

load ([name '.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);
alpha = thermcon / (density * specheat);
p = alpha * dt / dx^2;

% set initial conditions to 16C throughout.
% Do this for first two timesteps.
u([1 2], :) = 16 + 273.15;

% set up index vector
im = [2 1:nx-2];
```

```
i  = 1:nx-1;
ip = 2:nx;

% Assuming ambient temperature for time greater than 2000s
timedata(end:end+1) = [2008 2010];
tempdata(end:end+1) = 16 + 273.15;

% 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');
    u(n+1,nx) = R; % outer surface - boundary condition

    % Select method.
    switch method
        case 'forward'
            u(n+1,i) = (1 - 2 * p) * u(n,i) + p * (u(n,im) + u(n,ip));

        case 'dufort-frankel'
            u(n+1,i) = ((1 - 2 * p) * u(n-1,i) + 2 * p * (u(n,im) +
u(n,ip))) / (1 + 2 * p);

        case 'backward'
            b(1)          = 1 + 2 * p;
            c(1)          = -2 * p;
            d(1)          = u(n,1);
            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);
            a(nx)         = 0;
            b(nx)         = 1;
            d(nx)         = R;
            u(n+1,:)      = tdm(a,b,c,d);

        case 'crank-nicolson'
            b(1)          = 1 + p;
            c(1)          = -p;
            d(1)          = (1 - p) * u(n,1) + p * u(n,2);
            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);
            a(nx)         = 0;
            b(nx)         = 1;
            d(nx)         = R;
            u(n+1,:)      = tdm(a,b,c,d);

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

    end
end

if doplot
    % Create a plot here.
    % simple 3D plot
    surf(x,t,u);
```

```
    shading('interp')
    colorbar
    view(120,20)
    xlabel('\itx\rm (m)')
    ylabel('\itt\rm (s)')
    zlabel('\itu\rm (K)')
    xlim([0 xmax])
    ylim([0 tmax])
    zlim([0 1400])
    title(['Temperature profile' ' for ' name ' using ' method ' method'])

end
% End of shuttle function
```

Appendix B: MATLAB code for *tdm.m*

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

Appendix C: MATLAB code for *MethodCompare_nt.m*

```
% Plot the inner surface temperature variation with timestep
%
i=0;
nx = 21; % number of spatial steps
thick = 0.05; % total thickness
tmax = 4000; % maximum time
name = 597; % image to access

% Varies number of timesteps
for nt = 41:20:1001
    i=i+1;
    dt(i) = tmax/(nt-1);
    disp (['nt = ' num2str(nt) ', dt = ' num2str(dt(i)) ' s'])
    [~, ~, u] = shuttle(name, tmax, nt, thick, nx, 'forward', false);
    uf(i) = u(end, 1);
    [~, ~, u] = shuttle(name, tmax, nt, thick, nx, 'backward', false);
    ub(i) = u(end, 1);
    [~, ~, u] = shuttle(name, tmax, nt, thick, nx, 'dufort-frankel',
false);
    ud(i) = u(end, 1);
    [~, ~, u] = shuttle(name, tmax, nt, thick, nx, 'crank-nicolson',
false);
    uc(i) = u(end, 1);
end
semilogx(dt, [uf; ub; ud; uc])
ylim([430 440])
xlabel('\itdt\rm (s)')
ylabel('\itu\rm (K)')
title('Inner surface temperature variation with timestep')
legend ('Forward', 'Backward', 'Dufort-Frankel', 'Crank-
Nicolson','Location','northwest')
```

Appendix D: MATLAB code for *MethodCompare_nx.m*

```
function [xmax] = inSurfTemp(name,tmax,nt,nx,method)
% Function to plot the inner surface temperature variation with thickness
of tile
%
% Input arguments:
% name   - name of image to scan
% tmax   - maximum time
% nt     - number of timesteps
% nx     - number of spatial steps
% method - solution method ('forward', 'backward' etc)
%
% Return arguments:
% xmax   - optimum thickness

i = 0;

% Variation of thickness
thickMin = 0.05; % minimum thickness
thickMax = 0.07; % maximum thickness
thickChange = 0.001; % change in thickness

stop = 0;

% Varies total thickness
for thick = thickMin:thickChange:thickMax
    i = i + 1;
    % Find variation of inner surface temperature with time for thick
    [~,t,u] = shuttle(name, tmax,nt,thick,nx,method,false);
    innerU(i,:) = transpose( u(:,1) ); % vector of temperature variation
    thicknesses(i) = thick; % vector of different thicknesses

    % Detect index for thickness when inner surface temperature unchanged
    if ( ( ( u(:,1) - 450 ) < 1 ) & stop == 0 )
        optimumThick = i;
        stop = stop + 1;
    end

end

% plot of inner surface temperature against time  for range of thicknesses
h = surf(t,thicknesses,innerU);
set(h,'LineStyle','none');
view(120,20)
xlabel('\itt\rm (s)')
ylabel('\itthickness\rm (m)')
zlabel('\itu\rm (K)')
xlim([0 4000])
ylim([thickMin thickMax])
zlim([0 1400])
title(['Temperature profile' ' for temp' num2str(name) ' using ' method '
method'])

% Display suitable tile thickness based on inner surface temperature
xmax = thicknesses(optimumThick);
disp(['Suitable tile thickness is ' num2str(xmax) ' m'])
end
```

Appendix E: MATLAB code for *inSurfTemp.m*

```
function [xmax] = inSurfTemp(name,tmax,nt,nx,method)
% Function to plot the inner surface temperature variation with thickness
of tile
%
% Input arguments:
% name   - name of image to scan
% tmax   - maximum time
% nt     - number of timesteps
% nx     - number of spatial steps
% method - solution method ('forward', 'backward' etc)
%
% Return arguments:
% xmax   - optimum thickness

i = 0;

% Variation of thickness
thickMin = 0.05; % minimum thickness
thickMax = 0.07; % maximum thickness
thickChange = 0.001; % change in thickness

stop = 0;

% Varies total thickness
for thick = thickMin:thickChange:thickMax
    i = i + 1;
    % Find variation of inner surface temperature with time for thick
    [~,t,u] = shuttle(name, tmax,nt,thick,nx,method,false);
    innerU(i,:) = transpose( u(:,1) ); % vector of temperature variation
    thicknesses(i) = thick; % vector of different thicknesses

    % Detect index for thickness when inner surface temperature unchanged
    if ( ( ( u(:,1) - 450 ) < 1 ) & stop == 0 )
        optimumThick = i;
        stop = stop + 1;
    end

end

% plot of inner surface temperature against time  for range of thicknesses
h = surf(t,thicknesses,innerU);
set(h,'LineStyle','none');
view(120,20)
xlabel('\itt\rm (s)')
ylabel('\itthickness\rm (m)')
zlabel('\itu\rm (K)')
xlim([0 4000])
ylim([thickMin thickMax])
zlim([0 1400])
title(['Temperature profile' ' for temp' num2str(name) ' using ' method '
method'])

% Display suitable tile thickness based on inner surface temperature
xmax = thicknesses(optimumThick);
disp(['Suitable tile thickness is ' num2str(xmax) ' m'])
end
```

Appendix F: MATLAB code for *plottemp.m*

```
function [] = plottemp(name)
% function to plot image of measured temperature automatically.
%
%
%
img=imread([name '.jpg']);

% Create matrix indicating coordinate of colour
[height,width,~] = size(img);
Red = zeros(height,width);
Black = zeros(height,width);

Red( img(:,:,1)> 200 & img(:,:,2)< 100 & img(:,:,3)< 100 ) = 1;
Black( img(:,:,1)< 100 & img(:,:,2)< 100 & img(:,:,3)< 100 ) = 1;

% Find origin coordiantes
originY = find( sum(Black.') == max(sum(Black.')) );
originY = originY(1);
origin(2) = height - originY;
originX = find( sum(Black) == max(sum(Black)) );
origin(1) = originX(1);

% Find x = 2k coordinates
x2k = Black(originY,origin(1):end);
x2k = find( x2k == 1, 1, 'last');
x2k = x2k + origin(1) - 1;

% Find y = 2k coordinates
y2k = Black(1:originY,origin(1));
y2k = find( y2k == 0, 1,'last');
y2k = y2k + 1 ;
y2k = height - y2k;

% Find coordinates of red dots
[tempdata,timedata] = find ( Red == 1);
tempdata = height - tempdata;

% sort data and remove duplicate points.
[timedata, index] = unique(timedata);
tempdata = tempdata(index);

xScale = 2000 / (x2k - origin(1)); % x-coordinate scaling factor
yScale = 2000 / (y2k - origin(2)); % y-coordinate scaling factor

% Convert to time and temperature coordinates
timedata = (timedata - origin(1)) * xScale;
tempdata = (tempdata - origin(2)) * yScale;

% Convert temperature to Kelvin
tempdata = ((tempdata - 32) * (5/9)) + 273.15;

%save data to .mat file with same name as image file
save(name, 'timedata', 'tempdata')

end
```

Appendix G: MATLAB code for *shootingMethod.m*

```
function [reqThick] = shootingMethod(name)
% Calculate the thickness of tile required for different temperature plot
%
%
% set paramaters for shuttle.m function
tmax = 4000; % maximum time
nt = 801; % number of timesteps
nx = 34; % number of spatial steps
method = 'crank-nicolson'; % solution method
innerTemp = 450;

% 1st Guess
thick(1) = 0.05;
[~,~,u]=shuttle(name,tmax,nt,thick(1),nx,method,false);
intemp(1) = max( u(:,1) );
error(1) = innerTemp - intemp(1);

% 2nd Guess
thick(2) = 0.07;
[~,~,u]=shuttle(name,tmax,nt,thick(2),nx,method,false);
intemp(2) = max( u(:,1) );
error(2) = innerTemp - intemp(2);

i = 2;

while error(end) > 0.01

    i = i + 1;

    % Generate ith Guess
    thick(i) = thick(i-1) + error(end) * ((thick(end)-thick(end-
1))/(intemp(end)-intemp(end-1)));

    % Compute ith Guess
    [~,~,u]=shuttle(name,tmax,nt,thick(i),nx,method,false);
    intemp(i) = max( u(:,1) );
    error(i) = innerTemp - intemp(i);

end
reqThick = thick(i);
end
```

Appendix H: MATLAB code for *tilesThick.m*

```
% Calculate thickness of tile required for each temperature plot
%
location = [468 480 502 590 597 711 730 850];

for i = 1:8
    minThick = shootingMethod(location(i));
    disp(['Thickness of tile required for location ' num2str(location(i)) '
is ' num2str(minThick) ' m']);
end
```