Computer Project 2

10/12/2022

Team members	PSU ID	Contributions
Ankit Gupta	apg5667	Coding, Numerical Discretization
Aspen Stocking	abs6894	Analytical solution
Siddharth Premnath	sfp5478	Report writing, commenting the code
Nikhil Sharma	nvs5658	Numerical Discretization
Kameron Metcalf	kmm8076	Report Writing

Section 1: Analytical Solution

1) Analytical solution

$u_t = k u_{yy}$

$u_y(0,t) = -Q_1$

$u_y(H,t) = -Q_2$ } Non-Homogeneous

$u(y,0) = 0$

$V_t = kV_{yy}$

$V(y,0) = -\omega(y)$

$V_y(0,t) = V_y(H,t) = 0$

$F''(y) + P^2 F(y) = 0$

$G'(t) + kP_n^2 G(t) = 0$

$G(t) = e^{-kP_n^2 t}$

$u(y,t) = [\omega(y) + g(t)] + V(y,t)$

$u(y,t) - V(y,t) = \omega(y) + g(t)$

$u_t - V_t = g'$

$k(u_{yy} - v_{yy}) = k(\omega'')$

$g'(t) = k\omega''(y) = C$

$g(0) + \omega(y) = 0$

$g'(0) = -Q_1$

$g'(H) = -Q_2$

from table,

$F(y) = \begin{cases} A_0, & n=0 \\ A_n \cos(P_n y) \end{cases} \quad P_n = \frac{n\pi}{H}$

$V_n(y,t) = A_0 + A_n \cos(P_n y) e^{-kP_n^2 t}$

$V(y,t) = A_0 + \sum_{n=1}^{\infty} A_n \cos\left(\frac{n\pi}{H} y\right) e^{-k\left(\frac{n\pi}{H}\right)^2 t}$

$V(y,0) = -\omega(y) = A_0 + \sum_{n=1}^{\infty} A_n \cos\left(\frac{n\pi}{H} y\right)$

$A_0 = \frac{1}{H}\int_0^H -\omega(y)\, dy$

$A_n = \frac{2}{H}\int_0^H -\omega(y)\cos(P_n y)\, dy$

$$A_0 = \frac{-1}{H}\int_0^H \left(\frac{Q_1 - Q_2}{2H} y^2 - Q_1 y\right) dy$$

$$= -\frac{1}{H}\left(\frac{Q_1 - Q_2}{6H} y^3 - \frac{Q_1}{2} y^2\right)\Big|_0^H = -\frac{1}{H}\left(\frac{(Q_1 - Q_2)H^3}{6H} - \frac{3Q_1 H^3}{6H}\right)$$

$$= -\frac{1}{H}\left(\frac{(Q_1 - Q_2 - 3Q_1)H^3}{6H}\right) = -\frac{1}{H}\left(\frac{(-2Q_1 - Q_2)H^2}{6}\right)$$

$$= -\frac{H}{6}(-2Q_1 - Q_2) = \frac{H}{6}(Q_1 + Q_2)$$

$$A_n = \frac{2}{H}\int_0^H \left(\frac{Q_2 - Q_1}{2H} y^2 + Q_1 y\right)\cos(P_n y)\, dy$$

$$= \frac{2}{H}\left[\frac{Q_2 - Q_1}{2H}\left(\frac{2H\cos(P_n H)}{P_n^2} + \frac{P_n^2 H^2 - 2}{P_n^3}\sin(P_n H)\right)\right.$$

$$\left. + Q_1\left[\frac{1}{P_n^2}\cos(P_n H) + \frac{H}{P_n}\sin(P_n H)\right]\right]$$

$$= \frac{2}{H}\left[\frac{Q_2 - Q_1}{2H}\left(\frac{2H(-1)^n}{P_n^2}\right) + Q_1 \frac{(-1 + (-1)^n)H^2}{n^2\pi^2}\right]$$

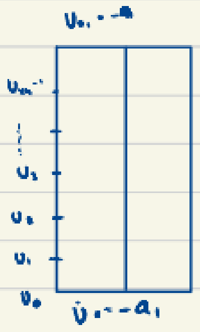
$$= \frac{2H}{n^2\pi^2}\left[Q_2(-1)^n - Q_1\right]$$

$$u(y,t) = [\omega(y) + g(t)] + A_0 + \sum_{n=1}^{\infty} A_n \cos\left(\frac{n\pi}{H} y\right) e^{-k\left(\frac{n\pi}{H}\right)^2 t}$$

Section 2: Numerical Discretization

Part 1

2-1) $\dot{U} = AU + \bar{f}$



finding boundary conditions

$$U_x(0,t) = \frac{-3U_0 + 4U_1 - 2U_2}{2\Delta Y} = -Q_1$$

$$U_0 = \frac{2\Delta Y \, Q_1}{3} + \frac{4}{3}U_1 - \frac{1}{3}U_2$$

$$U_y(H,t) = \frac{U_{N_y-2}(t) - 4U_{x-1}(t) + 3U_x(t)}{2\Delta Y} = -Q_2$$

$$U_{N_y} = -\frac{2\Delta Y \, Q_2}{3} + \frac{4}{3}U_{N_y-1} - \frac{1}{3}U_{N_y-2}(t)$$

Developing system of equations by explicit euler

$$U_{YY\,i} = \frac{U_{i-1} - 2U_i + U_{i+1}}{(\Delta Y)^2} \qquad 1 \leq i \leq N_y - 1$$

boundary cases

$$U_{YY\,1} = \frac{U_0 - 2U_1 + U_2}{(\Delta Y)^2} = \frac{1}{\Delta Y^2}\left[-\frac{2}{3}U_1 + \frac{2}{3}U_2\right] + \frac{2}{3\Delta Y}a_1$$

$$U_{YY\,N_y-1} = \frac{U_{N_y-2} - 2N_{y-1} + U_{N_y}}{\Delta Y^2}$$

$$= \frac{1}{\Delta Y^2}\left[\frac{2}{3}U_{N_y-2} - \frac{2}{3}U_{N_y-1}\right] - \frac{2}{3\Delta Y}(Q_2)$$

$\dot{U}$ conversion

$$\dot{U}_i = U_{t_i} = k\, U_{yy_i}$$

$$\dot{U}_1 = \frac{k}{\Delta y^2}\left[-\frac{2}{3}U_1 + \frac{2}{3}U_2\right] + \frac{2}{3\Delta y}a_1$$

$$\dot{U}_2 = \frac{k}{\Delta y^2}\left[U_1 - 2U_2 + U_3\right]$$

$$\vdots$$

$$\dot{U}_{N_y-2} = \frac{k}{\Delta y^2}\left[U_{N_y-3} - 2U_{N_y-2} + U_{N_y-1}\right]$$

$$\dot{U}_{N_y-1} = \frac{k}{\Delta y^2}\left[\frac{2}{3}U_{N_y-2} - \frac{2}{3}U_{N_y-1}\right] - \frac{2}{3\Delta y}(a_2)$$

Matrix form

$$\begin{bmatrix} \dot{U}_1 \\ \dot{U}_2 \\ \vdots \\ \dot{U}_{N_y-2} \\ \dot{U}_{N_y-1} \end{bmatrix} = \frac{k}{\Delta y^2}\begin{bmatrix} -2/3 & 2/3 & 0 & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 2/3 & -2/3 \end{bmatrix}\begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_{N_y-2} \\ U_{N_y-1} \end{bmatrix} + \frac{2}{3\Delta y}\begin{bmatrix} a_1 \\ 0 \\ \vdots \\ 0 \\ -a_2 \end{bmatrix}$$

$$\dot{U} = AU + f$$

Part 2

2-2) Crank Nicolson

① $$U_t = \frac{U_{j+1} - U_j}{\Delta t} = \dot{U}$$

② ① (i,j) $$U_{yy}^{(1)} = \frac{U(i, i-1) - 2U_{i,j} + (U_j, i+1)}{\Delta y^2}$$

② $(i+i, j)$ $$U_{yy}^{(2)} = \frac{U_{j+1, i-1} - 2U_{i+1, j} - U_{j+1, i+1}}{\Delta y^2}$$

$$U_{yy} = \frac{1}{2}\left(U_{yy}^{(1)} + U_{yy}^{(2)}\right)$$

$$U_{yy}^{2} = \frac{1}{2}\left(U_{yy}^{(1)} + U_{yy}^{(2)}\right)$$

$$U_t^{20} = k\, U_{yy}^{\omega}$$

$$r = \frac{\Delta t \cdot k}{\Delta y^2}$$

$$\frac{U_{j+1, i} - U_{i,1}}{\Delta t} = \frac{k}{2}\left(\frac{U(j, i-1) - 2U_{j,i} + (U_j, i+1)}{\Delta y^2} + \frac{U_{j+1\, i-1} - 2U_{(j+1)\, i} - U_{j+1\, i+1}}{\Delta y^2}\right)$$

$$(-r)\, U_{(i+1, i-1)} + 2(i+r)\, U_{(j+1, i)} - r(U_{j+1, i+1}$$

$$= r\, U_{(i, i+1)} + 2(1-r)\, U_{(i,j)} + r(U_{j, i+1})$$

Writing as a matrix

$$\underbrace{\begin{bmatrix} 1+\frac{1}{3}r & -\frac{1}{3}r & & \\ -\frac{1}{2}r & 1+r & -\frac{1}{2}r & \\ \ddots & \ddots & \ddots & \\ -\frac{1}{2}r & 1+r & -\frac{1}{2}r & \\ & -\frac{1}{3}r & 1+\frac{1}{3}r & \end{bmatrix}}_{T} \begin{bmatrix} U \end{bmatrix}_{n+1} = \underbrace{\begin{bmatrix} 1-\frac{1}{3}r & \frac{1}{3}r & & \\ \frac{1}{2}r & 1-r & \frac{1}{2}r & \\ \ddots & \ddots & \ddots & \\ \frac{1}{2}r & 1-r & \frac{1}{2}r & \\ & \frac{1}{3}r & 1-\frac{1}{3}r & \end{bmatrix}}_{S} \begin{bmatrix} U \end{bmatrix}_{n}$$

$$+ \underbrace{\begin{bmatrix} \Delta t \cdot \frac{2}{3\Delta y} Q_1 \\ 0 \\ \vdots \\ 0 \\ -\Delta t \cdot \frac{2}{3\Delta y} Q_2 \end{bmatrix}}_{f}$$

$$T = \begin{bmatrix} 1+\frac{1}{3}r & -\frac{1}{3}r & & \\ -\frac{1}{2}r & 1+r & -\frac{1}{2}r & \\ \ddots & & & \\ -\frac{1}{2}r & 1+r & -\frac{1}{2}r & \\ & -\frac{1}{3}r & 1+\frac{1}{3}r & \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & & 1 \end{bmatrix} - \frac{\Delta t}{2} \cdot \frac{k}{\Delta y^2} \begin{bmatrix} -2/3 & 2/3 & 0 \\ 1 & -2 & 1 \\ \ddots & \ddots & \ddots \\ 1 & -2 & 1 \\ & 2/3 & -2/3 \end{bmatrix}$$

$$= I - \frac{\Delta t}{2} A$$

$$S = \begin{bmatrix} 1+\frac{1}{3}r & -\frac{1}{3}r & & \\ -\frac{1}{2}r & 1+r & -\frac{1}{2}r & \\ \ddots & & & \\ -\frac{1}{2}r & 1+r & -\frac{1}{2}r & \\ & -\frac{1}{3}r & 1+\frac{1}{3}r & \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & & 1 \end{bmatrix} - \frac{\Delta t}{2} \cdot \frac{k}{\Delta y^2} \begin{bmatrix} -2/3 & 2/3 & 0 \\ 1 & -2 & 1 \\ \ddots & \ddots & \ddots \\ 1 & -2 & 1 \\ & 2/3 & -2/3 \end{bmatrix}$$

$$= I + \frac{\Delta t}{2} A$$

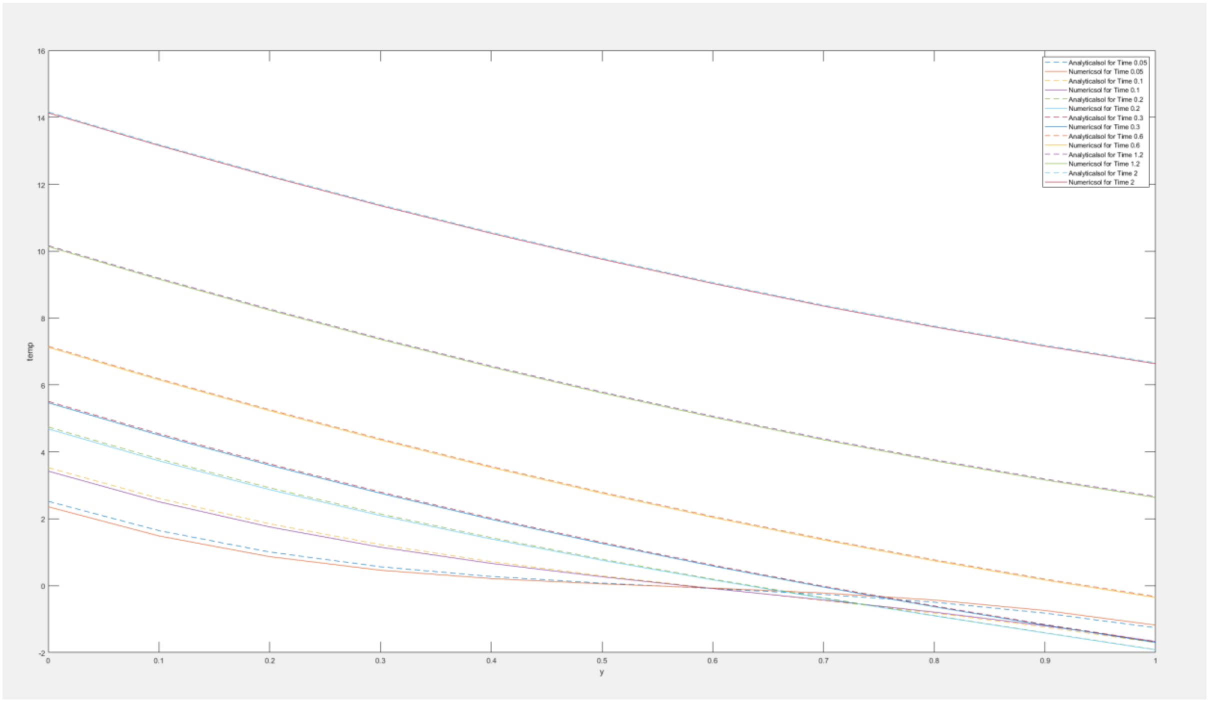
$$F = \begin{bmatrix} \Delta t \cdot \frac{2}{3\Delta y} Q_1 \\ 0 \\ \vdots \\ 0 \\ -\Delta t \cdot \frac{2}{3\Delta y} Q_2 \end{bmatrix} = \Delta t \begin{bmatrix} \frac{2}{3\Delta y} Q_1 \\ 0 \\ \vdots \\ 0 \\ \frac{-2}{3\Delta y} Q_2 \end{bmatrix} = \Delta t \cdot f$$

Section 4: Numerical Solutions and Discussion

Part 1

In the code

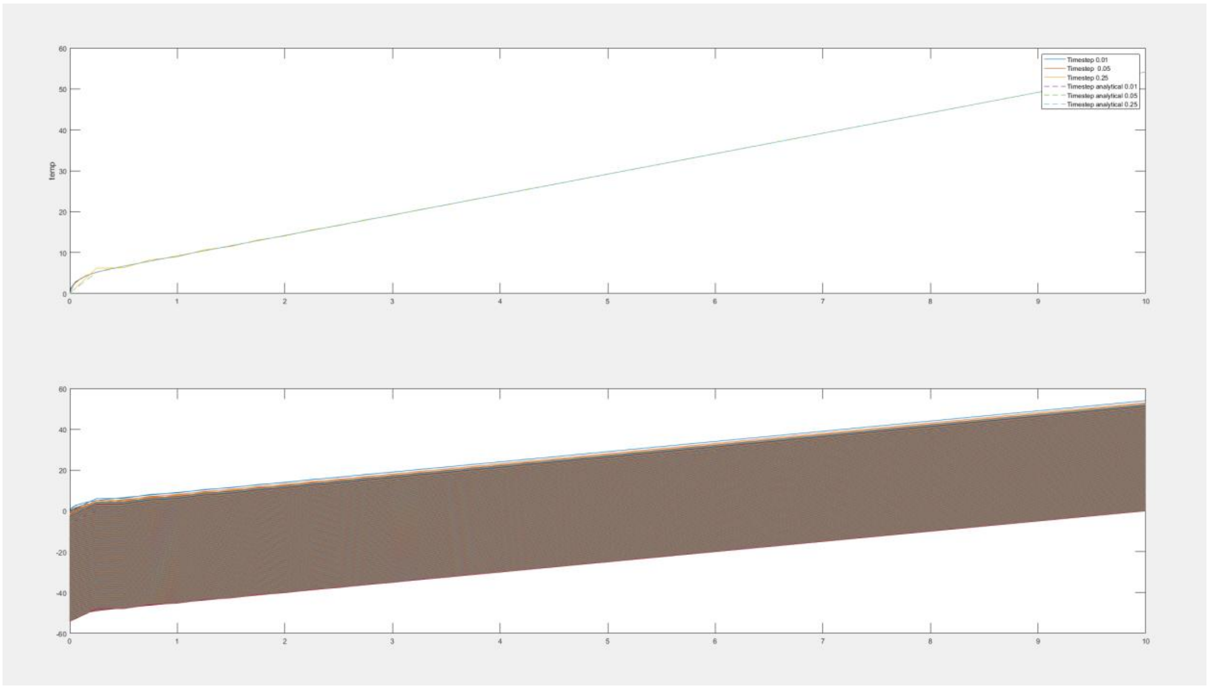
Part 2



Temperature vs Y

The plot above shows very little deviation from the analytical to the numerical solution and it can be found that as the time increases the temperature also increases throughout the length.

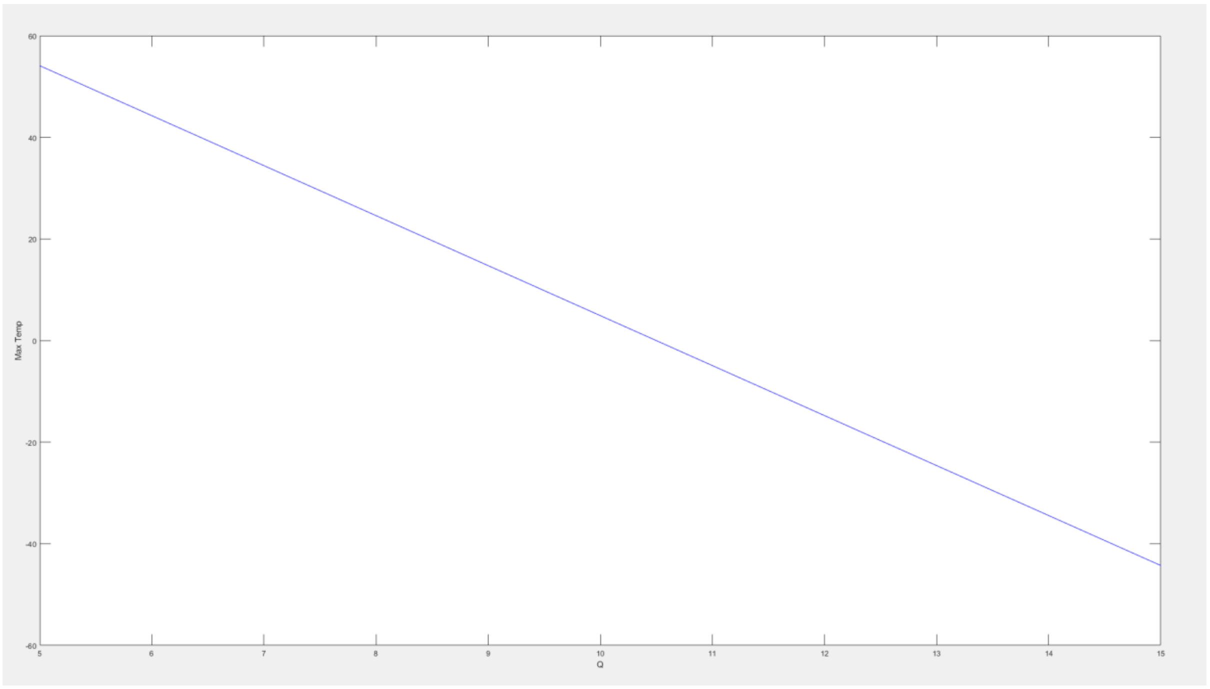
Part 3



Temperature vs Time

From the above graph it can be observed that as the delta time increases the more the deviation increases from the numerical and analytical curve indicating that the smaller time step is much more accurate in this analysis.

Part 4



Max Temperature vs Q

From the above graph we can see that when Umax=5 the corresponding Q value is 10 and based on these conditions with q2 =10, we can design a new system to achieve these conditions.

Section 5: Conclusion

Like stated in the problem statement, the goal of this project is to find out which value of Q_2 so that the maximum temperature increase of the structure is less than the prescribed value u_{max} during the entire mission profile of $0 < t \leq T$. Thankfully after some reduction and modeling the equations used for this project come out to be

$$u_t = \kappa(u)u_{yy}$$
$$u(0,y) = 0$$
$$u_y(0,t) = -Q_1\big(u(o,t)\big)$$
$$u_y(H,t) = Q_2$$

Where u is the temperature increase, κ is the coefficient of the structure's properties, Q_1 is the aerodynamic heating, $y = 0$ and Q_2 is the heat flux that is absorbed by the cooling system at $y = H$.

Analytical Solutions:
For the analytical solution, we were able to determine that we can split Eq. (1) into

$$u(y,t) = [w(y) + g(t)] + v(y,t)$$

Where $w(y) + g(t)$ form

$$w(y) = \frac{Q_1 - Q_2}{2H} y^2 - Q_1 y$$
$$g(t) = \frac{(Q_1 - Q_2)\kappa}{H} t$$

And that $v(y,t)$ satisfies this PDE

$$v_t = \kappa(u) v_{yy}$$
$$v(0,y) = -w(y)$$
$$v_y(0,t) = 0$$
$$v_y(H,t) = 0$$

And for our complete solution we were able to calculate the following equation

$$u(y,t) = w(y) + g(t) + A_0 + \sum_{n=1}^{N} A_n \cos(p_n y) \exp\left(-\kappa p_n^2 t\right)$$

Where $p_n = \frac{n\pi}{H}$, and

$$A_0 = \frac{H}{6}(2Q_1 + Q_2)$$
$$A_n = \frac{2H}{n^2\pi^2}[(-1)^n Q_2 - Q_1]$$

Numerical Discretization
Like in the analytical section κ and Q_1. Unlike in class, where space and time were discretized simultaneously, here space is discretized first. This results in the first order ODE. This ODE was able to be discretized in time using a new method we learned called the Crank-Nicolson scheme. Both methods would result in the same answer. We were able to solve this part of the project with some relative ease.

Computer Program
For the computer program we used recitation as a template and made a few tweaks to it, resulting in our final code block. We would then use this code to solve the questions from section 4 Numerical Solutions and Discussion. We created graphs comparing our numerical and analytical solutions. And designed the cooling system so that we could solve our original problem. That being find out which value of Q_2 so that the maximum temperature increase of the structure is less than the prescribed value u_{max} during the entire mission profile of $0 < t \leq T$. The Code could have been improved with higher level graphing tools to analyze the solutions we got to a better extent. Also, the solution matrixes for smaller step sizes are very huge and displaying such large matrices is impossible.

Final Thoughts
Computer Project 2 for this class proved to be a difficult task. Assignments for this class and others, along with finals week on the horizon meant that we had to coordinate our efforts in such a manner that we could get this done. None the less, the effort put into this project proved to be worthwhile as time management was the biggest obstacle to overcome. But that was not enough to suede the group from completing the project. Help from the recitations and office hours with the professors was massive help, as the project was not easy in the slightest. The analytical part proved to be a challenge, as that required the most attention from the

group, and we had little understanding of how to start the problem. But like with the rest of the project it was finished.

Appendix: The Computer Program

```
%CP2
clc;
clear all;
close all;

%  -----------------------------
%  Constants
%  -----------------------------
kp = 1.0;     % kappa
h  = 1.0;
Ti = 5.0;
Q1 = 10.0;
Q2_ref = 5.0;
n = 12;

% ----------------------------------------
% Verification for Thomas algorithm
% ----------------------------------------
A =[2 -1  0  0 ;1  2 -1  0. ;0  1  2 -1 ;0  0  1  2];
d = [0.0 6.0 2.0 -5.0];
[a, b, c] = LUdecomp(A);
[x] = LUsolve(a, b, c, d);


% ----------------------------------------
% (4)(1)  This is how the heatSolver is used.
% ----------------------------------------
dts = [0.01 0.05 0.25];
[y1, t1, u1] = heatSolver(0.1, dts(1), Q2_ref); %using the implimented Heatsolver
[y2, t2, u2] = heatSolver(0.1, dts(2), Q2_ref);%using the implimented Heatsolver
[y3, t3, u3] = heatSolver(0.1, dts(3), Q2_ref);%using the implimented Heatsolver


% ----------------------------------------
% (4)(2) For you to figure out
% ----------------------------------------
Ns=[5,10,20,30,60,120,200];
for i=1:length(Ns)
    t=Ns(i)*0.01;
    plot(y1,anaSol(y1,t,n,Q2_ref),"--",'DisplayName',['Analyticalsol for Time ' 
num2str(t)])
    hold on
    plot(y1,u1(Ns(i),:) ,'DisplayName',['Numericsol for Time ' num2str(t)])

end
xlabel("y");
ylabel("temp");
legend();
%--
%the plot above shows very little diviation from the analytical to the
%numerical solution and it can be found that as the time increases the
% temperature also increases throughout the beam
```

```matlab
% ----------------------------------------
% (4)(3) For you to figure out
% ----------------------------------------
figure()
subplot(2,1,1);
plot(t1,u1(:,1),'-','DisplayName',['Timestep ' num2str(0.01)]);
hold on
plot(t2,u2(:,1),'-','DisplayName',['Timestep  ' num2str(0.05)]);
plot(t3,u3(:,1),'DisplayName',['Timestep ' num2str(0.25)]);
plot(t1,anaSol(0,t1,n,Q2_ref),'--','DisplayName',['Timestepanalytical '
num2str(0.01)]);
plot(t2,anaSol(0,t2,n,Q2_ref),'--','DisplayName',['Timestepanalytical '
num2str(0.05)]);
plot(t3,anaSol(0,t3,n,Q2_ref),'--','DisplayName',['Timestepanalytical '
num2str(0.25)]);
ylabel("temp")
legend();

subplot(2,1,2)
plot(t1,u1(:,1)-anaSol(0,t1,n,Q2_ref),'-');
hold on
plot(t2,u2(:,1)-anaSol(0,t1,n,Q2_ref),'-');
plot(t3,u3(:,1)-anaSol(0,t1,n,Q2_ref),'-');
%from the above graph it can be observed that as the delta time increases
%the more the deviasion increases from the numerical and analytical curve
%indicating that the smaller time step is much more accurate in this
%analysis.

% ----------------------------------------
% (4)(4) For you to figure out
% ----------------------------------------

Qs = linspace(5,15,31);
us=[];
for i=1:length(Qs)
    [~,~,u]= heatSolver(0.1,0.5,Qs(i));
    us=[us;u(length(u),1)];
end

figure()
plot(Qs,us,'b-','DisplayName',"Numerical Solution");
xlabel("Q");
ylabel("Max Temp");
% from the above graph we can see that the  when umax=5 the corresponding Q
% value is 10 and based on these conditions with q2 =10, we can design a
% new system to achive these conditions.

function [F, T, S, B1, B2] = initProblem(Ny, dy, dt, Q2)
%     """
%     A helper function called by *heatSolver* that prepares the quantities in Eq.
(11)
%     Input:
%     Ny: Number of steps in y-direction, an integer
%     dy: Step size in y-direction, a float
%     dt: Time step size, a float
%     Q2: Heat flux of the cooling system, default Q2=5.0
%     Output:
%     F: The forcing vector, (_Ny-1)-element vector
%     T: The LHS matrix, (_Ny-1)x(_Ny-1) matrix
%     S: The RHS matrix, (_Ny-1)x(_Ny-1) matrix
%     B1: Coefficients for computing u(0,t), 3-element vector
%     B2: Coefficients for computing u(1,t), 3-element vector
%     """

Q1=10;
nu=1;
N=Ny-1;
f=zeros(N,1);
f(1)=Q1*2/3;
f(N)=-Q2*2/3;
A= -2*diag(ones(1,N),0)+diag(ones(1,N-1),1)+diag(ones(1,N-1),-1);
A0=zeros(1,N);
A0(1)=-2/3;
A0(2)=-A0(1);
A(1,:)=A0;
A(N,:)=fliplr(A0);
I=diag(ones(1,N),0);
r=(nu/2)*(dt/dy^2);
F=dt/dy*nu*f;
T=I-r*A;
S=I+r*A;

% ----------------------------------------
% TODO: Construct the F, T, S arrays
% ----------------------------------------
% F = ???
% T = ???
% S = ???

% B vectors - already provided here
B1 =[Q1*(2*dy)/3.0 4.0/3.0 -1.0/3.0];
B2 =[-Q2*(2*dy)/3.0 4.0/3.0 -1.0/3.0];
end

function [a, b, c] = LUdecomp(T)
%     LU decomposition of a tridiagonal matrix in the form of three arrays.
%     Input:
%     T:  Tridiagonal matrix to decompose, NxN matrix
%     Output:
%     a: Main diagonal of U matrix, N-element array
%     b: Lower diagonal of L matrix, (N-1)-element array
%     c: Upper diagonal of U matrix, (N-1)-element array


a = diag(T);      % Initialize a by main diagonal of T
b = diag(T, -1);  % Initialize b by lower diagonal of T
c = diag(T, 1);   % Initialize c by upper diagonal of T
N = length(a);
for i=2:N
    % ----------------------------------------
    % TODO: Complete the loop to compute arrays of a and b
    % ----------------------------------------
    b(i-1)=b(i-1)/a(i-1);
    a(i)=a(i)-b(i-1)*c(i-1);
end

end
```

```
function [x] = LUsolve(a, b, c, d)
%     Solve a linear system LUx=b using backward substitution.
%     Input:
%     a, b, c: Output from LUdecomp
%     d: The RHS term of the linear system, N-element array
%     Output:
%     x: Solution, N-element array

x = zeros(size(d),'like',d);   % Initialize the solution array
N = length(d);
% ----------------------------------------
% TODO: Implement the backward substitution.
% ----------------------------------------
x(1)=d(1);
for j=2:N
    x(j)=d(j)-b(j-1)*x(j-1);
    x(N)=x(N)/a(N);
end
for k=N-1:-1:1
    x(k)=(x(k)-c(k)*x(k+1))/a(k);
end

end

function [y,t,U] = heatSolver(dy, dt, Q2)
%     Solves the unsteady heat transfer problem.
%     Input:
%     dy: Step size in y-direction, a float
%     dt: Time step size, a float
%     Q2: Heat flux of the cooling system, default Q2=5.0
%     Output:
%     y: Grid points in y-direction, (Ny+1)-element vector
%     t: All the time steps, (Nt+1)-element vector
%     U: An array containing all solutions, (_Nt+1)x(_Ny+1) matrix

% ----------------------------------------
% TODO: Comment on the lines below or develop your own implementation.
% ----------------------------------------

h  = 1.0;
Ti  = 10.0;

Ny = int16(ceil(h/dy));  % Determine the number of grid points
Nt = int16(ceil(Ti/dt));  % Determine the number of time steps
y  = linspace(0, h, Ny+1);  % Generate the grid point vector
t  = linspace(0, Ti, Nt+1);  % Generate the time step vector
U  = zeros(Nt+1, Ny+1);      % Allocate the array for numerical solutions

% Initialize the numerical discretization
[F, T, S, B1, B2] = initProblem(Ny, dy, dt, Q2);
% LU decomposition of the T matrix
[a, b, c] = LUdecomp(T);

for i = 1:Nt
    u = U(i,2:Ny)'; %Filling intermidiate matrix
    U(i+1,2:Ny) = LUsolve(a, b, c, S*u+F);
    U(i+1,1) = B1(1) + B1(2)*U(i+1,2) + B1(3)*U(i+1,3);
    U(i+1,Ny+1) = B2(1) + B2(2)*U(i+1, length(U(i+1,:))-1) + B2(3)*U(i+1,
length(U(i+1,:))-2);
end

end

function [u] = anaSol(y, t, n, Q2)
%     Generates analytical solution
%     Input:
%     y: The grid points at which the analytical solutions are evaluated, N-
element vector
%     t: The time at which the analytical solutions are evaluated, a float
%     n: Number of eigenfunctions to use, an integer
%     Q2: Heat flux of the cooling system
%     Output:
%     u: Analytical solutions evaluated at grid points *y* and time *t*, N-element
vector
nu=1.0;
h=1;
Q1=10;
u =  zeros(size(y),'like',y);
% ----------------------------------------
% TODO: Implement the analytical solution
% ----------------------------------------
w= (Q1-Q2)/(2*h)*y.^2-Q1*y;
u=u+w;
u=u+(2*Q1+Q2)*h/6;
for i=1 :n+1
    p=i*pi/h;
    if mod(i,2)==0
        An=(2*h*(Q2-Q1))/(i*pi)^2;
    else
        An=(2*h*(-Q1-Q2))/(i*pi)^2;
    end
    u=u+An*cos(p*y)*exp(-nu*p^2*t);
end
u=u+nu*(Q1-Q2)/h*t;
end
```