AERSP 313 Computer Project 2

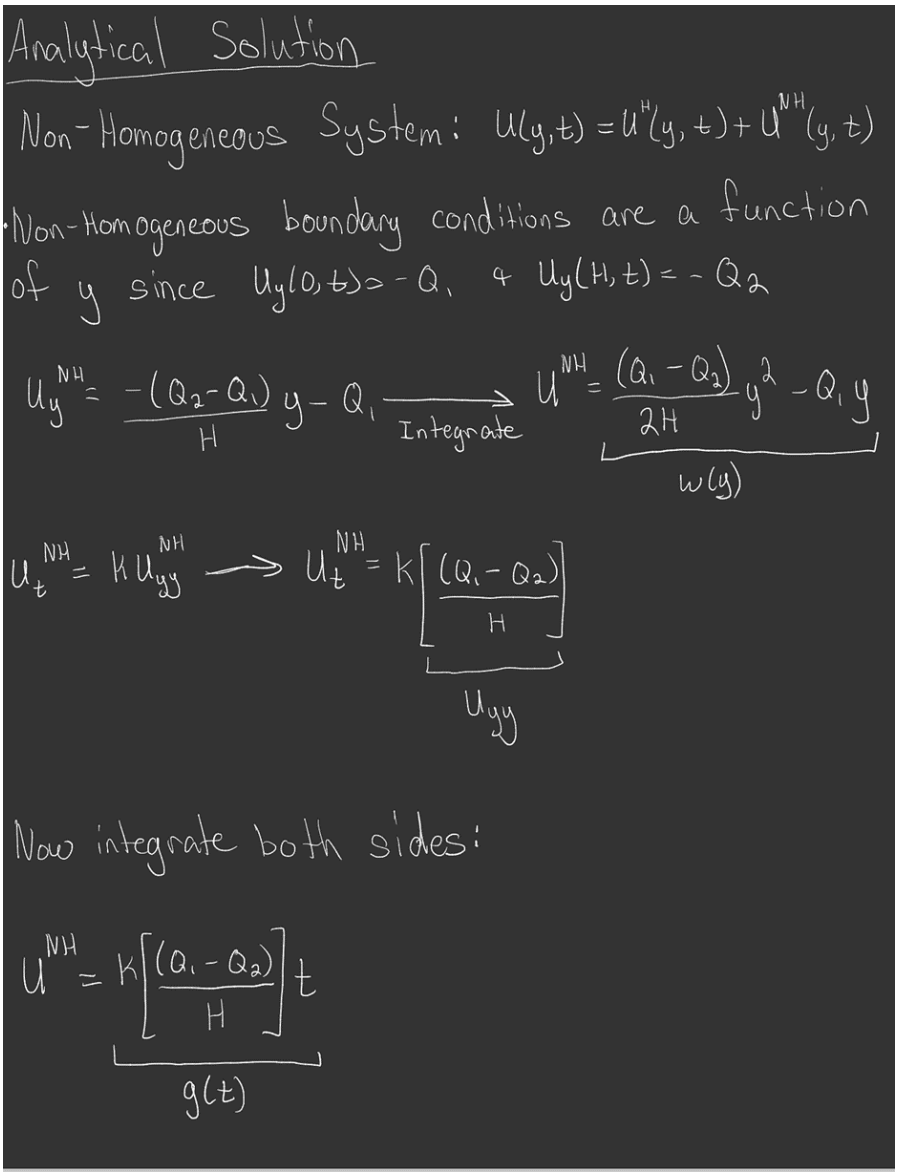
12/9/22

Team Members	PSU ID	Contributions
Jordan Sprague	jas8432	Numerical Discretize
Craig Stenstrom	tcs5426	Code
Jayden Slotnick	jcs6472	Analytical
Payton Glynn	pjg5446	Handwriting, Analytical
Nicholas Giampetro	njg5479	Code

Problem Definition

The purpose of this report is to determine a value for Q2 in which we ensure that the temperature increase the structure experiences is less than U_{max}. This value is under consideration for the entirety of the experimental duration from t>0 to t=T. In this project we use PDE solution verification, discretization, and computer code to solve the PDE and evaluate different time stamps. Our analysis of a PDE dependent on time and space will help us understand how to handle real-world test models and any compromising factors to structures over time.

Section 1: Analytical Solutions



Homogeneous Solution

$V_t = K V_{yy}$

$V(y,0) = -w(y)$

$V_y(0,t) = 0$

$V_y(H,t) = 0$

$U^{Total} = U^H + U^{NH}$

where $U^H = V(t,y)$ & $U^{NH} = w(y) + g(t)$

$$U^{Total}(y,t) = \underbrace{[w(y) + g(t)]}_{\text{Non-Homogeneous}} + \underbrace{V(t,y)}_{\text{Homogeneous}}$$

Now solve for $V(y,t)$:

$V_t = K V_{yy}$

From equation sheet: $F''(y) + K F(y) = 0$

$G'(t) + K C^2 G(t) = 0$

BC's: $V(y,t) = F(y)\,G(t)$

$0 = F'(0)\,G(t) \rightarrow F'(0) = 0$

$0 = F'(H)\,G(t) \rightarrow F'(H) = 0$

From table: $K_n = p_n^2 = \left(\frac{n\pi}{H}\right)^2$, $n = 0, 1, 2, 3 \ldots$

$F(y) = A_n \cos(p_n y)$

When $n = 0 \rightarrow F(y) = A_0$

When $n = 1, 2, 3 \ldots \rightarrow F(y) = A_n \cos(p_n y)$

Now find $G(t)$: $G'(t) + K c^2 G(t) = 0$

$G'(t) + p_n K^2 G(t) = 0$

$G'(t) = -p_n K^2 G(t)$

$G(t) = \exp(-p_n K^2 t)$

$$V_n(y,t) = A_0 + A_n \cos(p_n y) \exp(-p_n k^2 t)$$

$$V(y,t) = A_0 + \sum_{n=1}^{\infty} A_n \cos(p_n y) \exp(-p_n k^2 t)$$

Apply IC: $V(y,0) = -w(y)$

$$V(y,0) = -w(y) = A_0 + \sum_{n=1}^{\infty} A_n \cos(p_n y)$$

Fourier Series

$$A_0 = \frac{1}{2L}\int_{-L}^{L} f(x)dx = \frac{1}{H}\int_0^H -w(y)\,dy$$

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

$$A_0 = \frac{1}{H}\left[\frac{Q_2 - Q_1}{6H}y^3\Big|_0^H + \frac{Q_1 y^2}{2}\Big|_0^H\right]$$

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

$$A_0 = \frac{Q_2 H}{6} - \frac{Q_1 H}{6} + \frac{Q_1 H}{2}$$

$$A_0 = \frac{Q_2 H}{6} + \frac{Q_1 H}{3} = \frac{H(Q_2 + 2Q_1)}{6} \rightarrow \boxed{A_0 = \frac{H}{6}(2Q_1 + Q_2)}$$

Now find A_n:

$$A_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos\left(\frac{n\pi x}{L}\right)dx = \frac{1}{H}\int_{-H}^{H} -\omega(y)\cos(\rho_n y)\,dy$$

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

$$A_n = \frac{2}{H}\left[\frac{Q_2 - Q_1}{2H}\left[\frac{2y\cos(\rho_n y)}{\rho_n^2} + \frac{\rho_n^2 H^2 - 2}{\rho_n^3}\sin(\rho_n y)\right] + Q_1\left[\frac{1}{\rho_n^2}\cos(\rho_n y) + \frac{y}{\rho_n}\sin(\rho_n y)\right]\right]_0^H$$

$$A_n = \frac{2}{H}\left[\frac{Q_2 - Q_1}{2H}\left[\frac{2H\cos(\rho_n H)}{\rho_n^2} + \frac{\rho_n^2 H^2 - 2}{\rho_n^3}\sin(\rho_n H)\right] + Q_1\left[\frac{1}{\rho_n^2}\cos(\rho_n H) + \frac{H}{\rho_n}\sin(\rho_n H)\right]\right]$$

$$-\left[\frac{Q_2 - Q_1}{2H}(0) + 0\right] + Q_1\left[\frac{1}{\rho_n^2}\right]$$

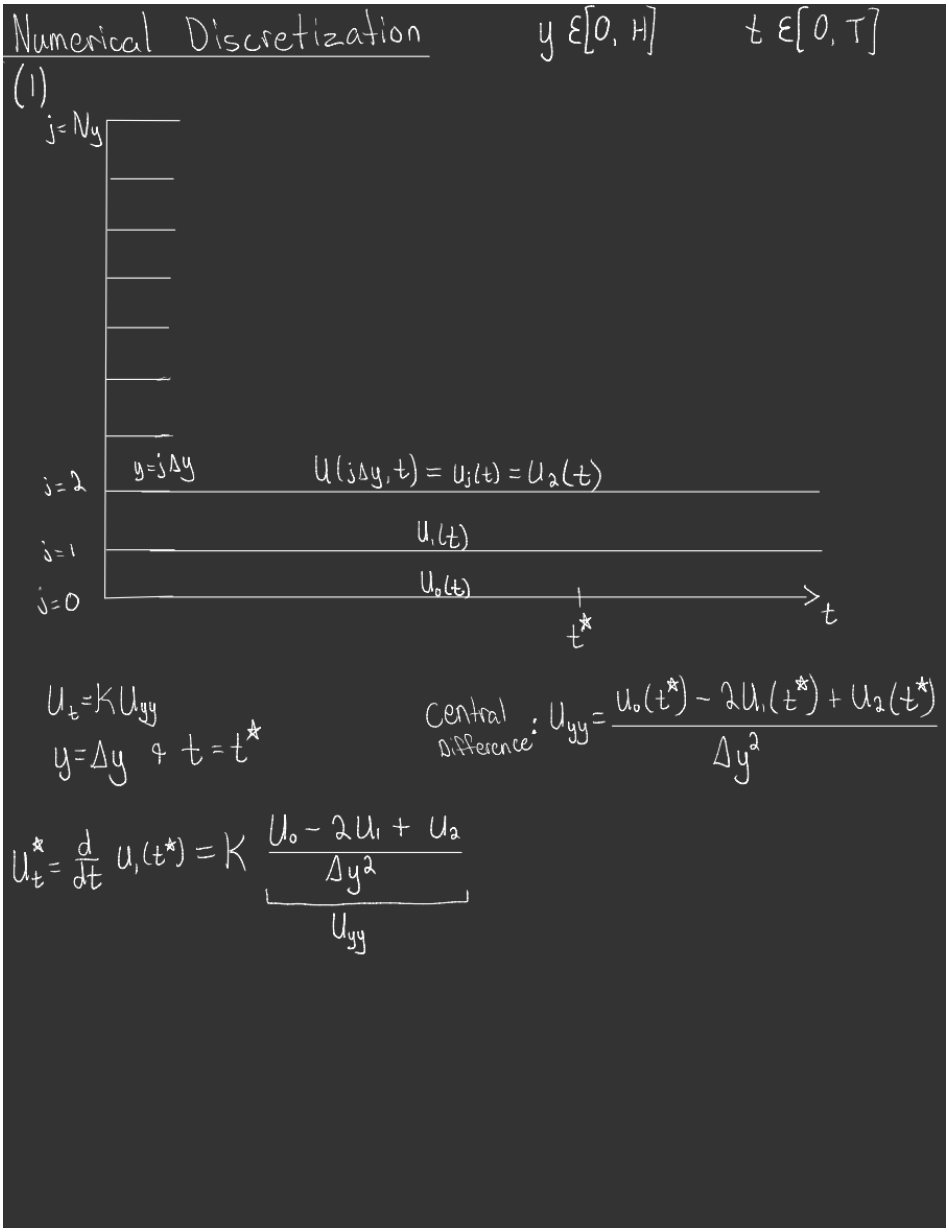
Note: $\rho_n \cdot H = n\pi$ and $\sin(n\pi) = 0$

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

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

Section 2: Numerical Discretization

Part 1:



$$\frac{d}{dt}\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix} = \frac{K}{\Delta y^2}\begin{bmatrix} -\frac{3}{3} & \frac{2}{3} & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & \frac{2}{3} & -\frac{2}{3} \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix} + \frac{2K}{\Delta y^2}\begin{bmatrix} Q_1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ -Q_2 \end{bmatrix}$$

$\dot{u}$ $\qquad$ A $\qquad$ u $\qquad$ $\tilde{f}$

Given Equations: $U_y(0,t) = \dfrac{-3u_0 + 4u_1 - u_2}{2\Delta y} = -Q_1$

$$\rightarrow u_0(t) = \frac{2\Delta y Q_1}{3} + \frac{4}{3}u_1(t) - \frac{1}{3}u_2(t)$$

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

$$\rightarrow U_{N_y-2} - 4U_{N_y-1} + 3U_{N_y} = -2\Delta y Q_2 \rightarrow 3U_{N_y} = -2\Delta y Q_2 + 4U_{N_y-1} - U_{N_y-2}$$

$$\rightarrow U_{N_y} = -\frac{2}{3}\Delta y Q_2 + \frac{4}{3}U_{N_y-1} - \frac{1}{3}U_{N_y-2}$$

Boundary Conditions: $U_y(0,t) = -Q_1$, $U_y(H,t) = -Q_2$

$$\frac{d}{dt}u_1 = \frac{K}{\Delta y^2}(u_0 - 2u_1 + u_2) = \frac{K}{\Delta y^2}\left[\left(\frac{2\Delta y Q_1}{3} + \frac{4}{3}u_1 - \frac{1}{3}u_2\right) - 2u_1 + u_2\right]$$

$$= \frac{K}{\Delta y^2}\left[\frac{2\Delta y Q_1}{3} - \frac{2}{3}u_1 + \frac{2}{3}u_2\right] = \frac{K}{\Delta y^2}\left(-\frac{2}{3}u_1 + \frac{2}{3}u_2\right) + \frac{2KQ_1}{3\Delta y}$$

$\tilde{f}_1$

So $\boxed{\dot{u}_1 = \frac{K}{\Delta y^2}\left(-\frac{2}{3}u_1 + \frac{2}{3}u_2\right) + \tilde{f}_1}$

$$\frac{d}{dt}u_2 = \frac{K}{\Delta y^2}(u_1 - 2u_2 + u_3)$$

plug into matrix above (2nd row)

$$\dot{u}_2 = \frac{K}{\Delta y^2}(u_1 - 2u_2 + u_3)$$

repeat this process for the remaining rows.

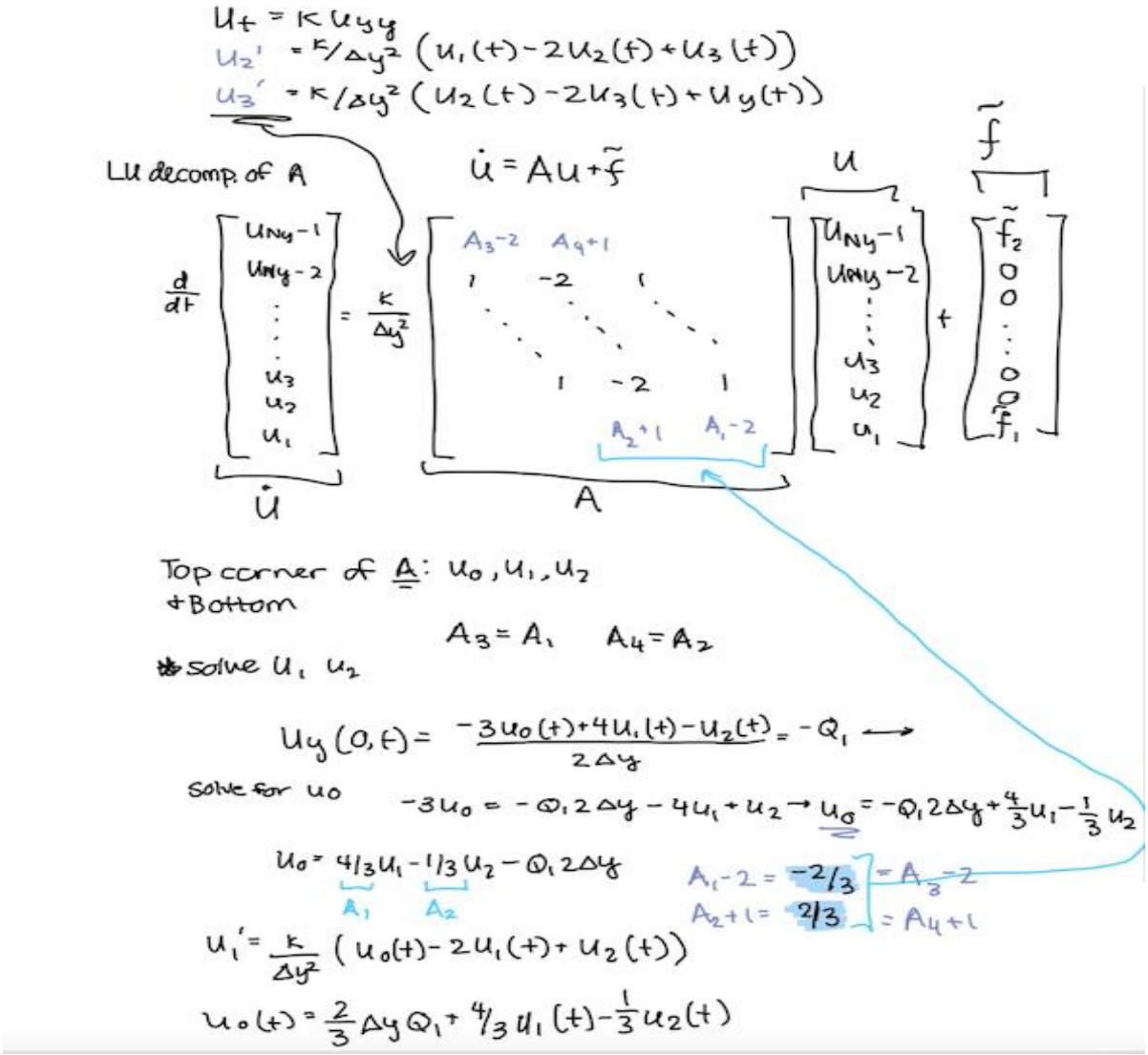
$$\frac{d}{dt}u_{Ny-1} = \frac{K}{\Delta y^2}\left(u_{Ny-2} - 2u_{Ny-1} + u_{Ny}\right)$$

$$\frac{d}{dt}u_{Ny-1} = \frac{K}{\Delta y^2}\left[u_{Ny-2} - 2u_{Ny-1} - \frac{2}{3}\Delta y Q_2 + \frac{4}{3}u_{Ny-1} - \frac{1}{3}u_{Ny-2}\right]$$

$$\frac{d}{dt}u_{Ny-1} = \frac{K}{\Delta y^2}\left[\frac{2}{3}u_{Ny-2} - \frac{2}{3}u_{Ny-1} - \frac{2}{3}\Delta y Q_2\right] = \frac{K}{\Delta y^2}\left(\frac{2}{3}u_{Ny-2} - \frac{2}{3}u_{Ny-1}\right) - \underbrace{\frac{2KQ_2}{3\Delta y}}_{\tilde{f}_2}$$

$$\dot{u}_{Ny-1} = \frac{K}{\Delta y^2}\left(\frac{2}{3}u_{Ny-2} - \frac{2}{3}u_{Ny-1}\right) - \frac{2KQ_2}{3\Delta y}$$

Part 2:



Top corner of $\underline{\underline{A}}$: u_0, u_1, u_2
+Bottom

$A_3 = A_1 \quad A_4 = A_2$

* solve u_1 u_2

$$u_y(0,t) = \frac{-3u_0(t) + 4u_1(t) - u_2(t)}{2\Delta y} = -Q_1 \longrightarrow$$

solve for u_0

$$-3u_0 = -Q_1 2\Delta y - 4u_1 + u_2 \rightarrow u_0 = -Q_1 2\Delta y + \frac{4}{3}u_1 - \frac{1}{3}u_2$$

$$u_0 = \underbrace{4/3}_{A_1} u_1 - \underbrace{1/3}_{A_2} u_2 - Q_1 2\Delta y$$

$A_1 - 2 = -2/3 = A_3 - 2$

$A_2 + 1 = 2/3 = A_4 + 1$

$$u_1' = \frac{K}{\Delta y^2}(u_0(t) - 2u_1(t) + u_2(t))$$

$$u_0(t) = \frac{2}{3}\Delta y Q_1 + 4/3\, u_1(t) - \frac{1}{3}u_2(t)$$

top part N_y-2 N_y-1 N_y

solve for u_{ny}

$u_1 \rightsquigarrow$ last term in front of a 0

then use $u_y(H,t)$

$$-Q_2 = \frac{u_{Ny-2} - 4u_{Ny-1} + 3u_{Ny}}{2\Delta y}$$

$$-\frac{2\Delta y}{3}Q_2 + \frac{4}{3}u_{Ny-1} - \frac{1}{3}u_{Ny-2} = u_{Ny}$$

$\tilde{f} \rightarrow \quad Q_1 \rightarrow -Q_2$

$$u_y(H,t) = \frac{u_{Ny-2}(t) - 4u_{Ny-1}(t) + 3u_{Ny}(t)}{2\Delta y} = -Q_2$$

$$f_1 \rightarrow \frac{2}{3}\Delta y\,(Q_1) \leftarrow [u_1(t)$$

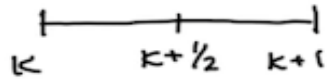
$\leftarrow$ common multiple in front of Q_1 & $-Q_2$ in $\tilde{f}$ vector

$$\tilde{f} = \frac{2k}{3\Delta y}\begin{bmatrix} Q_1 \\ \vdots \\ 0 \\ -Q_2 \end{bmatrix}$$

$$\frac{d}{dt}\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{Ny-2} \\ u_{Ny-1} \end{bmatrix} = \frac{k}{\Delta y^2}\begin{bmatrix} -2/3 & 2/3 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 2/3 & -2/3 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{Ny-2} \\ u_{Ny-1} \end{bmatrix} + \frac{2k}{3\Delta y}\begin{bmatrix} Q_1 \\ 0 \\ \vdots \\ 0 \\ -Q_2 \end{bmatrix}$$

Proof of $TU_{k+1} = SU_k + f$

For crank Nicolson: $k+1/2$



$$\dot{u}_{k+1/2} = A u_{k+1/2} + f$$

$$\dot{u}_{k+1/2} = \frac{u_{k+1} - u_k}{\Delta t} \qquad u_{k+1/2} = \frac{1}{2}(u_{k+1} + u_k)$$

$$\frac{u_{k+1} - u_k}{\Delta t} = A\left[\frac{1}{2}(u_{k+1} + u_k)\right] + \tilde{f}$$

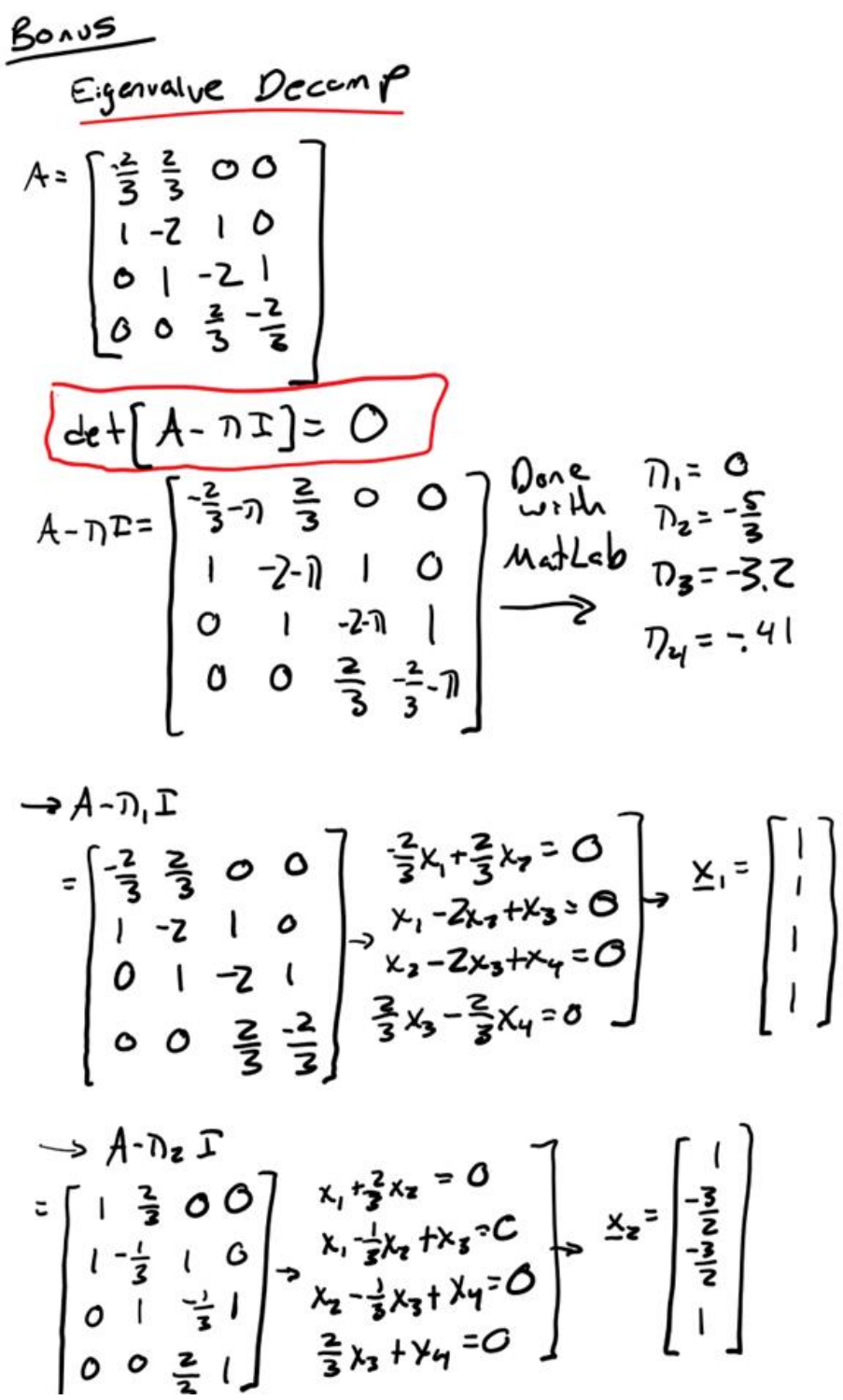
$$I u_{k+1} - I u_k = \frac{\Delta t}{2} A u_{k+1} + \frac{\Delta t}{2} A u_k + \tilde{f} \cdot \Delta t$$

$\tilde{f} = \Delta t \tilde{f}$

$$= \underbrace{\left(I - \frac{\Delta t}{2} A\right)}_{T} u_{k+1} = \underbrace{\left(I + \frac{\Delta t}{2} A\right)}_{S} u_{k-1} + \tilde{f} \Delta t$$

$\rightarrow$ $\boxed{TU_{k+1} = SU_k + f}$ ✓

Bonus


Bonus
Eigenvalue Decomp
A = [-2/3 2/3 0 0; 1 -2 1 0; 0 1 -2 1; 0 0 2/3 -2/3]
det[A - ηI] = 0
A - ηI = [-2/3-η 2/3 0 0; 1 -2-η 1 0; 0 1 -2-η 1; 0 0 2/3 -2/3-η]
Done with MatLab
η1 = 0
η2 = -5/3
η3 = -3.2
η4 = -.41
A - η1 I
-2/3 x1 + 2/3 x2 = 0
x1 - 2x2 + x3 = 0
x2 - 2x3 + x4 = 0
2/3 x3 - 2/3 x4 = 0
x1 = [1; 1; 1; 1]
A - η2 I
x1 + 2/3 x2 = 0
x1 - 1/3 x2 + x3 = 0
x2 - 1/3 x3 + x4 = 0
2/3 x3 + x4 = 0
x2 = [1; -3/2; -3/2; 1]

$\rightarrow A - D_3 I$

$$= \begin{bmatrix} 2.53 & \frac{2}{3} & 0 & 0 \\ 1 & 1.2 & 1 & 0 \\ 0 & 1 & 1.2 & 1 \\ 0 & 0 & \frac{2}{3} & 2.53 \end{bmatrix} \rightarrow \begin{matrix} 2.53x_1 + \frac{2}{3}x_2 = 0 \\ x_1 + 1.2x_2 + x_3 = 0 \\ x_2 + 1.2x_3 + x_4 = 0 \\ \frac{2}{3}x_3 + 2.53x_4 = 0 \end{matrix} \rightarrow \underline{x}_3 = \begin{bmatrix} -1 \\ 3.886 \\ -3.886 \\ 1 \end{bmatrix}$$

$\rightarrow A - D_4 I$

$$= \begin{bmatrix} -.25 & \frac{2}{3} & 0 & 0 \\ 1 & -1.59 & 1 & 0 \\ 0 & 1 & -1.59 & 1 \\ 0 & 0 & \frac{2}{3} & -.25 \end{bmatrix} \rightarrow \begin{matrix} -.25x_1 + \frac{2}{3}x_2 = 0 \\ x_1 - 1.59x_2 + x_3 = 0 \\ x_2 - 1.59x_3 + x_4 = 0 \\ \frac{2}{3}x_3 - .25x_4 = 0 \end{matrix} \rightarrow \underline{x}_4 = \begin{bmatrix} -1 \\ -.386 \\ .386 \\ 1 \end{bmatrix}$$

$$\underline{\underline{X}} = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & -\frac{3}{2} & 3.886 & -.386 \\ 1 & -\frac{3}{2} & -3.886 & .386 \\ 1 & 1 & 1 & 1 \end{bmatrix} \qquad \underline{\underline{X}}^{-1} = \begin{bmatrix} .3 & .2 & .2 & .3 \\ .2 & -.2 & .2 & .2 \\ -.045 & .117 & -.117 & .045 \\ -.455 & -.117 & .117 & .455 \end{bmatrix}$$

$$\underline{\underline{A}} = \underline{\underline{X}}\,\underline{\underline{\Lambda}}\,\underline{\underline{X}}^{-1}$$

$$\underline{\underline{A}} = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & -\frac{3}{2} & 3.886 & -.386 \\ 1 & -\frac{3}{2} & -3.886 & .386 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -\frac{5}{3} & 0 & 0 \\ 0 & 0 & -3.2 & 0 \\ 0 & 0 & 0 & -.41 \end{bmatrix} \begin{bmatrix} .3 & .2 & .2 & .3 \\ .2 & -.2 & .2 & .2 \\ -.045 & .117 & -.117 & .045 \\ -.455 & -.117 & .117 & .455 \end{bmatrix}$$

Section 3: Application of the Computer Program and Results

Results of numeric method in MATLAB

Figure 4 – Exact solution compared to numeric solution with time step of 0.01 at different times

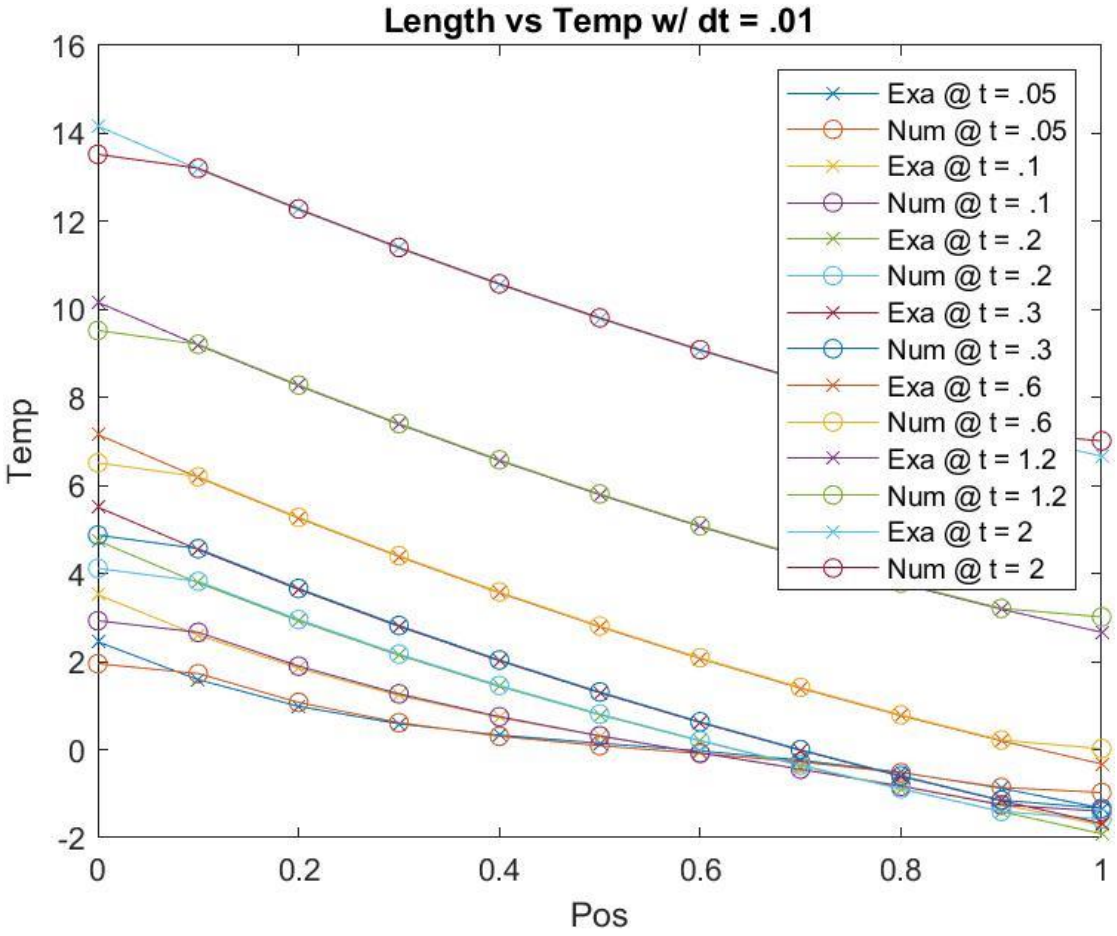


Figure 5 – Comparison of exact solution and numeric solution with different time steps at y = 0

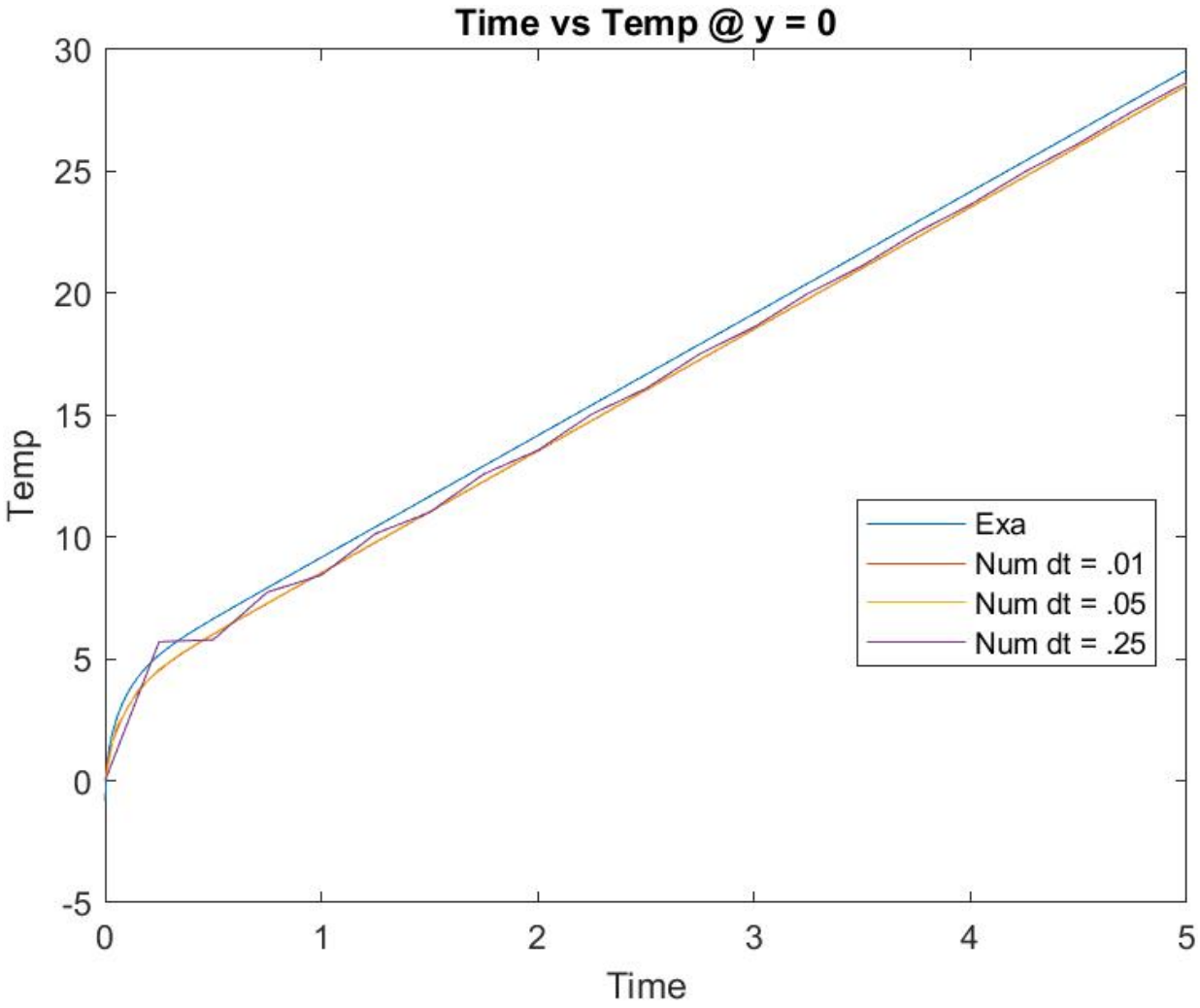
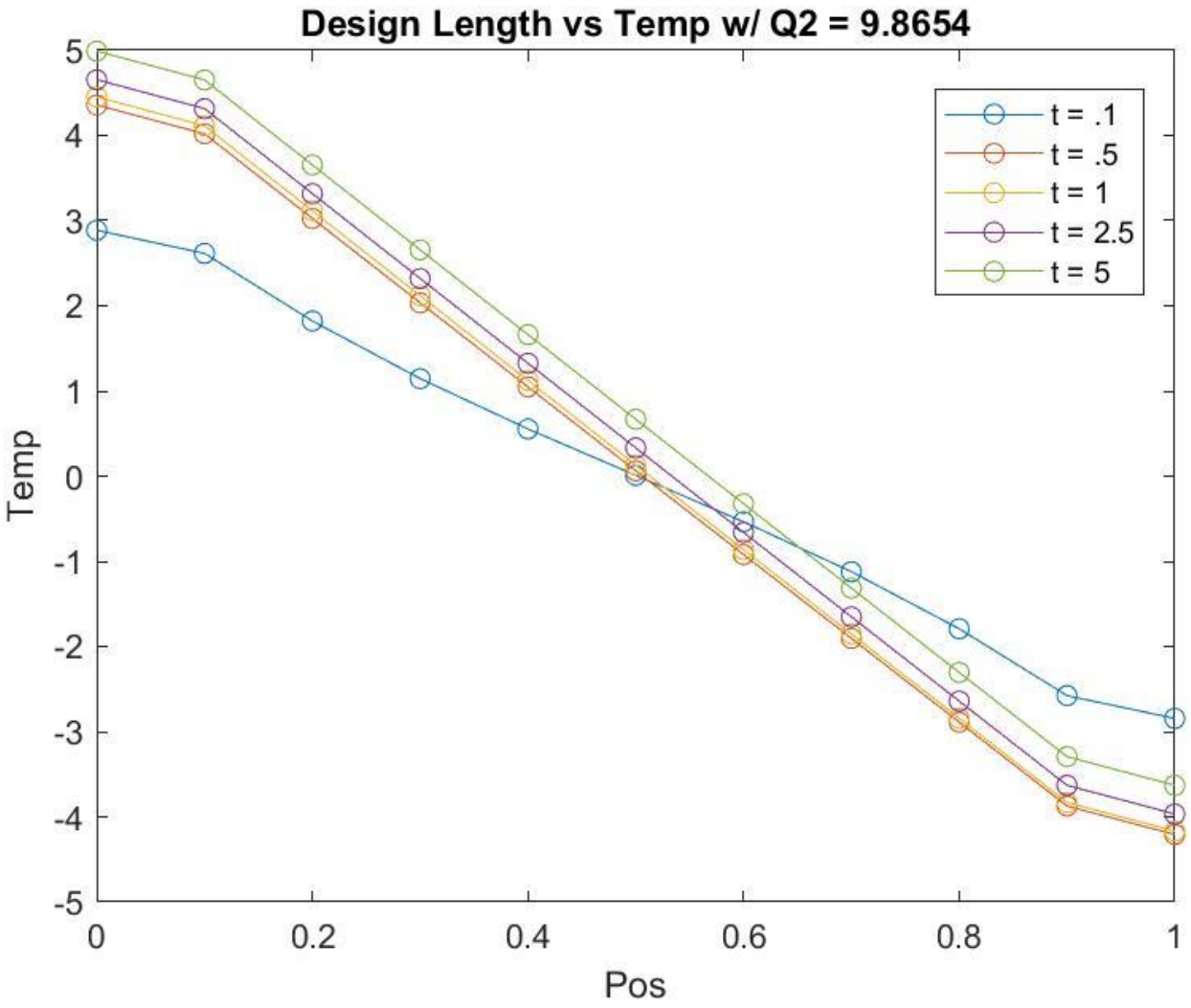


Figure 6 – Results of algorithm to find Q2 with a max temperature just less than 5 over 5 seconds



One trend that our group found interesting was that the results were off by a factor of the reciprocal of time step size which was 100 in this case. Then, when we multiplied the final results by the step size which was 0.01, the results fit nicely compared to the analytical. This correction worked for all cases including the general case. One other trend that we found interesting was that in Figure 4, the beginning and end points of the analytical and numerical solution were off by a significant margin, but the middle points had little to no difference. This trend shows up for all values of t.

Bonus

We attempted the final bonus but were unable to complete it with the correct graph. We have still included our code to show our attempt, and hopefully earn some credit.

Conclusions

The numerical solutions compared well with the analytical solutions. The results of the plot of Q2 verses time gave a nice indication of what we would have to pick our Q2 value to be in order to have the max temperature increase be less than the prescribed u_{max} value. The Q2 value we found to stay under u_{max} is Q2 = 9.8654, as shown in Figure 6. It is helpful to have the numerical solution as things become

nonlinear because an analytical solution is extremely difficult or impossible to achieve. The numerical method allows you to model increasingly complex situations. This project was helpful to all team members as we all learned a lot about analytical and numerical solving techniques which are very applicable to the aerospace industry. Also, attempting and completing some of the bonus problems forced us to expand our knowledge and think outside the box.

Appendix: Code

```
%AERSP 313 - CP2 - Matlab Code
%Code Skeleton Start
%by Jordan Sprague, Craig Stenstrom, Jayden Slotnick, Payton Glynn, Nicholas Giampetro



clc;
clear all;
close all;

%  ------------------------------
%  Constants
%  ------------------------------
setGlobalKappa(1.0);     % kappa
setGlobalH(1.0);
setGlobalTi(5.0);
setGlobalQ1(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);
x

% ---------------------------------------
% (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);
[y2, t2, u2] = heatSolver(0.1, dts(2), Q2_ref);
[y3, t3, u3] = heatSolver(0.1, dts(3), Q2_ref);

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

% initializing analyitical solution
anaSol1 = zeros(length(y1),1);

% loop to make plot with analyical and numeric solution at each time t
t = [.05,.1,.2,.3,.6,1.2,2];
```

```
for i = 1:length(t)
    for j = 1:11
        anaSol1(j) = anaSol(y1(j),t(i),1000,Q2_ref);
    end
    subplot(1,1,1)
    plot(y1,anaSol1,'-X')
    hold on
    plot(y1,u1(t(i)/dts(1)+1,:),'-o')
end
hold off

title('Length vs Temp w/ dt = .01')
xlabel('Pos')
ylabel('Temp')
legend('Exa @ t = .05','Num @ t = .05','Exa @ t = .1','Num @ t = .1','Exa @ t = .↙
2','Num @ t = .2','Exa @ t = .3','Num @ t = .3','Exa @ t = .6','Num @ t = .6','Exa @ t↙
= 1.2','Num @ t = 1.2','Exa @ t = 2','Num @ t = 2','location','best')

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

% initializing analytical solution
anaSol2 = zeros(length(t1),1);

% filling in analytical solution
for j = 1:length(t1)
anaSol2(j) = anaSol(0,t1(j),1000,Q2_ref);
end

% making plot with each of the time steps and analytical
figure
subplot(1,1,1)
plot(t1,anaSol2,'-')
hold on
plot(t1,u1(:,1),'-')
hold on
plot(t2,u2(:,1),'-')
hold on
plot(t3,u3(:,1),'-')
hold off

title('Time vs Temp @ y = 0')
xlabel('Time')
ylabel('Temp')
legend('Exa','Num dt = .01','Num dt = .05','Num dt = .25','location','best')

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

```
%initial guess
Q2_design = 5.0 ;
[yD, tD, uD] = heatSolver(0.1, dts(1), Q2_design);
maxTemp = max(max(uD)) ;

%loop to converge on a Q2 which results in a temp near 5
while ( maxTemp > 5) || ( maxTemp < 4.99)

    if maxTemp > 5
        Q2_design = Q2_design*2;
    elseif maxTemp < 4.9
        Q2_design = Q2_design/1.5;
    end

    [yD, tD, uD] = heatSolver(0.1, dts(1), Q2_design);
    maxTemp = max(max(uD));
end

% making a plot of the temps for the new Q2
figure
tD = [.1,.5,1,2.5,5];
for i = 1:length(tD)
    subplot(1,1,1)
    plot(yD,uD(tD(i)/dts(1)+1,:),'-o')
    hold on
end
hold off

title("Design Length vs Temp w/ Q2 = " + Q2_design)
xlabel('Pos')
ylabel('Temp')
legend('t = .1','t = .5','t = 1','t = 2.5','t = 5','location','best')

% ----------------------------------------
% (4)(5) Bonus
% ----------------------------------------

% Not sure if this answer makes any sense but I'm getting something


%initial guess
Q2_designBonus = 5 ;
[y1B, t1B, u1B] = heatSolverB(0.1, dts(1), Q2_designBonus);
maxTemp = max(max(u1B)) ;

% Never converges in current state
% loop to converge on a Q2 which results in a temp near 5
% while ( maxTemp > 5) || ( maxTemp < 4.99)
%
```

```
%     if maxTemp > 5
%         Q2_designBonus = Q2_designBonus*2;
%     elseif maxTemp < 4.9
%         Q2_designBonus = Q2_designBonus/1.5;
%     end
%
%     [y1B, t1B, u1B] = heatSolverB(0.1, dts(1), Q2_designBonus);
%     maxTemp = max(max(u1B));
% end

% making a plot of the temps for the new Q2
figure
t1B = [.1,.5,1,2.5,5];
for i = 1:length(t1B)
    subplot(1,1,1)
    plot(y1B,u1B(t1B(i)/dts(1)+1,:),'-o')
    hold on
end
hold off

title("Bonus graph, incomplete and wrong")
xlabel('Pos')
ylabel('Temp')
legend('t = .1','t = .5','t = 1','t = 2.5','t = 5','location','best')



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
%     """

% getting constants
Q1 = getQ1;
k = getKappa;

% N is the size of the return matricies
N = Ny-1;
```

```
% making the identity matrix
I  = diag(ones(N,1),0) ;

% solving for forcing vector C is just the value the matrix is multiplied by
F = zeros(N,1);
C = (2*k)/(3*dy);
F(1) = C*Q1;
F(N) = -C*Q2;

% creating A, the matrix which T and S will be found with C is again a value the matrix↙
gets multiplied by
A = diag(-2*ones(N,1),0) + diag(ones(N-1,1),1) + diag(ones(N-1,1),-1);
A(1,2)=2;
A(N,N-1)=2;
A(1,:) = A(1,:)/3;
A(N,:) = A(N,:)/3;
C = k/(dy^2);
A = C*A;

% solving for T and S according to formulas from question 2
T = I - (dt/2)*A;
S = I + (dt/2)*A;

% 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);   % Size of a diagonal

% creation of alpha beta and gamma vectors as an intermediate to solve a, b, and c
alpha = zeros(N,1);
beta = c ;
gamma = zeros(N,1);
gamma(1) = a(1);
```

```
% loop which finds a and b vectors
for i=2:N
    % equations to solve LUdecomp
    alpha(i) = b(i-1)/gamma(i-1);
    gamma(i) = a(i) - alpha(i)*beta(i-1);

    %sets a and b to their return values
    a(i) = gamma(i);
    b(i-1) = alpha(i);
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

N = length(d);  % size of return matrix

% creates the lower element matrix
L = diag(ones(N,1),0) + diag(b,-1) ;

% creates the upper element matrix
U = diag(a,0)+diag(c,1) ;

% solves the system of equations to return value x
y = L\d;
x = (U\y)';

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.
% ----------------------------------------
```

```
% getting the constants from the global constant functions
h = getH;
Ti = getTi;

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);

% loop to iterate through matrix and solve PDE at each time step
for i = 1:Nt
    u = U(i,2:Ny)';↙
% saving the Ny time step to u
    U(i+1,2:Ny) = LUsolve(a, b, c, S*u+F);↙
% applying formula from question 2 part 2 to approximate next time step
    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

% our results were always off by a factor of the reciporical of the dt
% this line corrects the results back into the correct size
U = U*dt;

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

% getting constants
h = getH;
Q1 = getQ1;
```

```
K = getKappa;

% initializing sum component of solution
sumComp = 0 ;

% solutions for A0 w(y) and g(t)
A0 = h/6*(2*Q1+Q2);
WY = ((Q1-Q2)*y^2)/(2*h) - Q1*y;
GT = ((Q1-Q2)*K*t)/h;

% finding an approximate value for the summation sum(An*cos(pny)exp(-k*pn^2*t)
for n=1:N
    Pn = ((n*pi)/h);
    An = ((2*h)/(n^2*pi^2))*((-1^n*Q2)-Q1);
    sumComp = sumComp + An*cos(Pn*y)*exp(-K*Pn^2*t);
end

% returning solution to u
u = WY + GT + A0 + sumComp;

end

%% part 4 bonus functions




function [F, A, B1, B2] = initProblemB(Ny, dy, 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
%     """

% getting constants
Q1 = getQ1;
k = getKappa;

% N is the size of the return matricies
N = Ny-1;
```

```
% making the identity matrix
I  = diag(ones(N,1),0);

% solving for forcing vector C is just the value the matrix is multiplied by
F = zeros(N,1);
C = (2*k)/(3*dy);
F(1) = C*Q1;
F(N) = -C*Q2;

% creating A, the matrix which T and S will be found with C is again a value the matrix↙
gets multiplied by
A = diag(-2*ones(N,1),0) + diag(ones(N-1,1),1) + diag(ones(N-1,1),-1);
A(1,2)=2;
A(N,N-1)=2;
A(1,:) = A(1,:)/3;
A(N,:) = A(N,:)/3;


% solving for T and S according to formulas from question 2


% 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 [y,t,U] = heatSolverB(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

% getting the constants from the global constant functions
h = getH;
Ti = getTi;

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, A, B1, B2] = initProblemB(Ny, dt, Q2);
```

```
N = Ny-1;

I  = diag(ones(N,1),0);


% loop to iterate through matrix and solve PDE at each time step
for i = 1:Nt
    u = U(i,2:Ny)';↙
% saving the Ny time step to u

    A = diag(-2*ones(N,1),0) + diag(ones(N-1,1),1) + diag(ones(N-1,1),-1);
    A(1,2)=2;
    A(N,N-1)=2;
    A(1,:) = A(1,:)*bonusKappa(U(i,1))/3;
    A(N,:) = A(N,:)*bonusKappa(U(i,N))/3;

    for j=2:N
        A(j-1,j-1)=A(j-1,j-1)*bonusKappa(U(i,j-1));
        A(j-1,j)=A(j-1,j)*bonusKappa(U(i,j-1));
        A(j,j-1)=A(j,j-1)*bonusKappa(U(i,j-1));
    end
    Q1=-(-3*U(i,1)+4*u(1)-u(2))/(2*dy)+.1*U(i,1);
    T = I - (dt/2)*A/dy^2;
    S = I + (dt/2)*A/dy^2;
    [a, b, c] = LUdecomp(T);
    F = zeros(N,1);

    F(1) = (2*bonusKappa(U(i,1)))/(3*dy)*(-getQ1+.1*U(i,1));
    F(N) = -(2*bonusKappa(U(i,N)))/(3*dy)*Q2;



    U(i+1,2:Ny) = LUsolve(a, b, c, S*u+F);↙
% applying formula from question 2 part 2 to approximate next time step
    B1 =[Q1*(2*dy)/3.0 4.0/3.0 -1.0/3.0];
    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

% our results were always off by a factor of the reciporical of the dt
% this line corrects the results back into the correct size
U = U*dt;

end

function [k] = bonusKappa(u)
```

```
k=1-.1*u;

end

%% Global variable functions to make the consts easier to work with
% they make MATLAB angry but make me happy so MATLAB can cry about it

function setGlobalKappa(val)
global kp
kp = val;
end

function r = getKappa
global kp
r = kp;
end

function setGlobalH(val)
global h
h = val;
end

function r = getH
global h
r = h;
end

function setGlobalTi(val)
global Ti;
Ti = val;
end

function r = getTi
global Ti
r = Ti;
end

function setGlobalQ1(val)
global Q1
Q1 = val;
end

function r = getQ1
global Q1
r = Q1;
end
```