Project 3: Feedback Temperature Control

Nicholas McLaughlin

MECH.5410 – Advanced Heat Transfer

Professor Juan Pablo Trelles

November 10th, 2020

Table of Contents

Variables Used

Below is a complete list of the variables and notations used throughout this analysis.

Variable	Description	Value/Equation	Units
L_x	Length of the surface along x	0.020	[m]
L_y	Length of the surface along y	0.010	[m]
R	Radius of the circular quadrant heating element	0.005	[m]
t_{final}	Final time to be considered	100	[s]
ρC_p	Volumetric heat capacity	2000	[J/m^3/K]
k	Thermal conductivity	1.0	[W/m^2/K]
T_∞	Ambient temperature	$10 + 10\sin\left(\frac{9\pi t}{t_{final}}\right) + 7\sin\left(\frac{22\pi t}{t_{final}}\right) + 5\sin\left(\frac{7\pi t}{t_{final}}\right)$	[˚C]
T_{inital}	Initial temperature	10	[˚C]
T_r	Reference temperature	20	[˚C]
g	Heat generation	Controlled Parameter	[J/m^3]
T	Temperature	Variable	[˚C]
N_x	Spatial nodes along x	51	[ ]
N_y	Spatial nodes along y	25	[ ]
N	Total number of nodes	$N_x \times N_y$	[ ]
i	Index along y	Variable	[ ]
j	Index along x	Variable	[ ]
$\boldsymbol{T}$	Discretized temperature vector	See Equation 3	[˚C]
$\boldsymbol{A}$	System matrix	See Code in Appendix	[ ]
$\boldsymbol{B}$	Input matrix	See Section 1.6	[ ]
$\boldsymbol{u}$	Control vector	See Equation 4.1	[ ]
$\boldsymbol{y}$	Output vector	See Equation 4.2	[˚C]
$\boldsymbol{C}$	Output matrix	See Section 1.7	[ ]
$\boldsymbol{K}$	Gain matrix	See Equation 4.4	[ ]
$\boldsymbol{r}$	Reference vector	See Equation 4.3	[˚C]
δt	Desired time interval	User defined	[s]
k_g	Heat generation gain	Variable	[ ]

1 – Thermal Model

1.1 – Schematic Model

This analysis considers the efficacy of a surface-temperature control system for various thermocouple configurations and gain coefficients. A schematic of this, for the general case, is shown below in Figure 1.

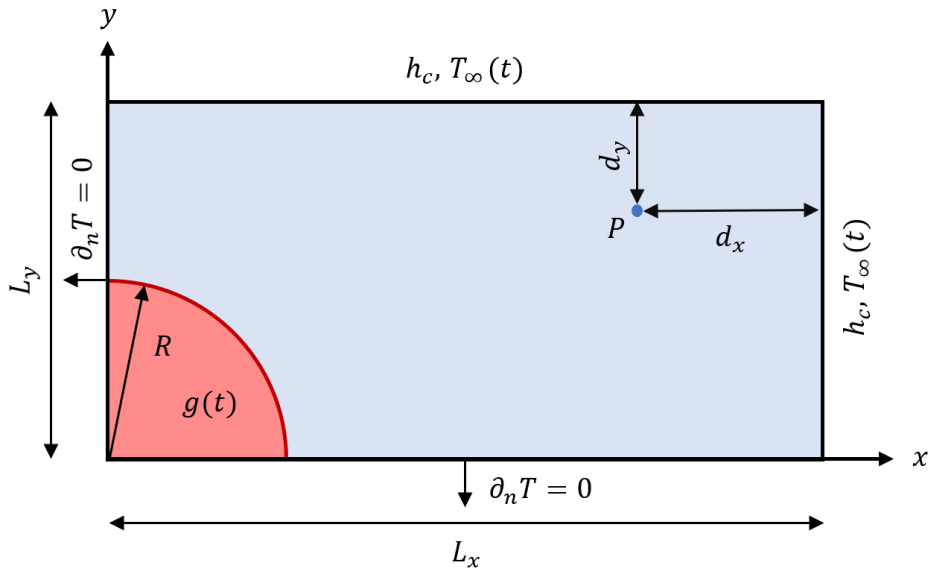


Figure 1. Schematic Diagram of Surface of Interest

The control system is used to regulate the temperature at the top and right boundaries such that there is a minimal deviation from a specified reference temperature. The control system uses a heater in the shape of a circular quadrant (lower-left corner) to regulate the temperatures along these boundaries.

Previously, a control system was developed to consider the system in Figure 1 when $d_x = d_y = 0$, and the right boundary is under the same conditions as the left and bottom boundaries. Therefore, a slight modification needs to be made to the existing controls to account for variable d_x and d_y as well as the convective right boundary condition.

1.2 – Background Theory

The heat transfer considerations within this analysis are based on the "linear constant-properties heat conduction equation" as shown below in Equation 1.

$$\frac{\partial T}{\partial t} = \frac{k}{\rho C_p}\nabla^2 T + \frac{g}{\rho C_p} \tag{1}$$

This equation fundamentally describes the heat transfer phenomena occurring on the surface. For

1.3 – Discretizing the Domain

If the surface is to be discretized into a grid N_x by N_y, each cell can be assigned an index value. Index $i = 1$ begins at the origin and increases along x until $i = N_x$. Then, $i = N_x + 1$ begins 1 row above and continues the pattern. This continues until $i = N = N_x$ x N_y in the top right corner.

An example of this indexing for N_x = 20 and N_y = 10 is provided below in Figure 2.

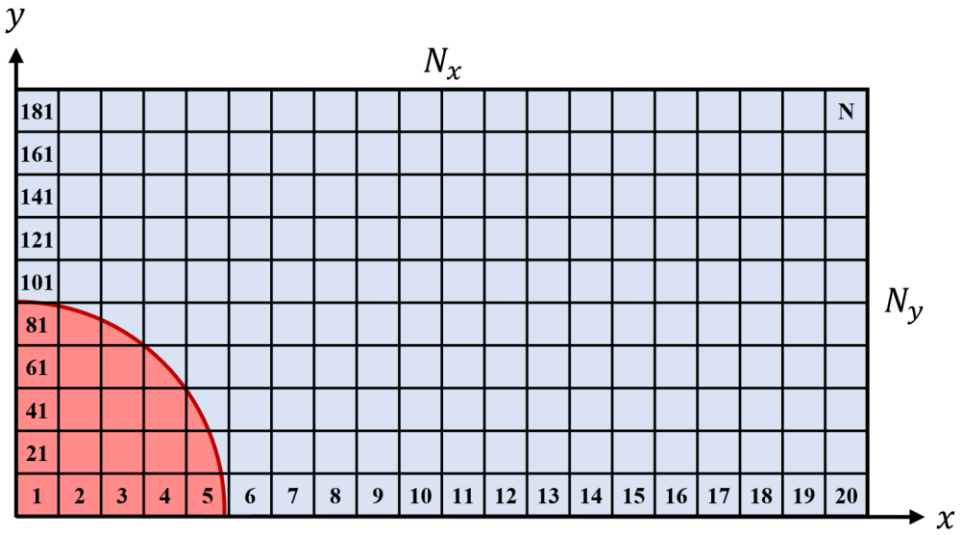


Figure 2. Discretized Surface for N_x = 20 and N_y = 10

1.4 – Converting between Cartesian Coordinates and Indices

Any 2D Cartesian coordinate requires a transformation into the indices used to discretize the area. In MATLAB, this transformation is given by the following code.

```
Px = round(Nx*(x/Lx));
Py = round(Ny*(y/Ly));
Node = Nx * ( Py - 1 ) + Px;
```

Effectively, this converts the desired position x and y into a single value where the point in the *x*-*y* plane would approximately be on the discretized grid.

1.5 – State-Space Formulation

A general system with feedback control and a reference input can be described by the following equations [1].

$$\dot{\boldsymbol{T}} = \boldsymbol{AT} + \boldsymbol{Bu} \tag{2.1}$$

$$\boldsymbol{y} = \boldsymbol{CT} \tag{2.2}$$

$$\boldsymbol{u} = \boldsymbol{K}(\boldsymbol{r} - \boldsymbol{y}) \tag{2.3}$$

In these equations $\boldsymbol{T}$ is the discretized temperature vector, $\boldsymbol{A}$ is the system matrix, $\boldsymbol{B}$ is the input matrix, $\boldsymbol{u}$ is the control vector, $\boldsymbol{y}$ the output vector, $\boldsymbol{C}$ the output matrix, $\boldsymbol{K}$ the gain matrix, and $\boldsymbol{r}$ the reference vector [1]. The temperature at a specific time interval, δt, is provided below in Equation 3 [1].

$$\boldsymbol{T}^{n+1} = [\boldsymbol{I} - \delta t(\boldsymbol{A} - \boldsymbol{BCK})]^{-1}[\boldsymbol{T}^{n} + \delta t\boldsymbol{BKr}] \tag{3}$$

The matrix $\boldsymbol{k}$ and the vectors $\boldsymbol{u}$, $\boldsymbol{y}$, and $\boldsymbol{r}$ used in the previous equations are further defined in Equations 4.1-4.4.

$$\boldsymbol{u} = \begin{Bmatrix} g \\ T_\infty \end{Bmatrix} \tag{4.1}$$

$$\boldsymbol{y} = \begin{Bmatrix} T_p \\ 0 \end{Bmatrix} \tag{4.2}$$

$$\boldsymbol{r} = \begin{Bmatrix} T_r \\ T_\infty \end{Bmatrix} \tag{4.3}$$

$$\boldsymbol{k} = \begin{bmatrix} k_g & 0 \\ 0 & 1 \end{bmatrix} \tag{4.4}$$

1.6 – The Input Matrix

The input matrix, $\boldsymbol{B}$, depends on the geometry and boundary conditions of the surface. In general, it is a size N×2 matrix filled with zeros. Only at special locations is $b_{ij} \neq 0$. These locations are defined as follows:

$b_{i1} = \frac{1}{\rho C_p}$ when the i^{th} node corresponds to the geometry generating heat

$b_{i2} = -h$ when the i^{th} node corresponds to the convective boundaries

Else, $b_{ij} = 0$

To consider the boundaries in the index notation, the locations can be considered in the matrix discretization first then converted into the index form. For example, the top boundary occurs in the matrix discretization as the entire last row. Each position in the last row (i.e., $N_{xi} = 1{:}N_x$ and $N_{yi} = N_y$) can be converted into the index form with the MATLAB code provided in 1.3 and the boundary conditions stored in context. An example of this for the top boundary is provided below:

```
i = Ny;
for j = 2 : Nx-1
    node  = Nx * ( i - 1 ) + j;
    nodeb = node - Nx;
    A( node, node  ) = vln * ( -k / dy - h );
    A( node, nodeb ) = vln * (  k / dy     );
    B( node, 2     ) = vln * (  h          );
end
```

For the entire code considering each boundary, see the Appendix.

The locations of the heat generation are a bit more involved to evaluate. The positions in the discretized matrix need to be checked if they are within the area of the circular quadrant then they can be stored in context. The code for this is provided below.

```
for i = 2 : Ny-1
    for j = 2 : Nx-1
        node = Nx * ( i - 1 ) + j;
        if sqrt( x( j )^2 + y( i )^2 ) <= R
            B( node, 1 ) = 1.0 / rhoCp;
        end
    end
end
```

1.7 – The Output Matrix

The output matrix, ***C***, is a size 2×N matrix filled with zeros. Only when the following condition is satisfied does $c_{ij} \neq 0$

$c_{1j} = 1$ when the j^{th} node corresponds to the point where the thermocouple is

Else, $c_{ij} = 0$

This is not trivial – this process requires the index transformation provided in 1.3 to consider the point in context. This code is repeated below.

```
Pnode = Nx * ( Py - 1 ) + Px;
Ctx.node = Pnode;
C = sparse(2, N);
C(1, Ctx.node) =  1;
```

2 – Numerical Solution

2.1 – Temperature Distribution as a Function of Thermocouple Position

To consider the temperature distribution resulting from the surface-temperature control system, a code in MATLAB was developed to evaluate the linear system described in Equations 2 and 3. This code can be found in Appendix A.2. The majority of the functional code comes from Professor Juan Pablo Trelles' code "heatplate_feedcontrol.m". Professor Trelles' code evaluates the linear system over a prescribed time range and plots the surface temperature distribution for several instants as well as the controls response. The thermocouple is always considered to be in the top right corner and only the top boundary undergoes convection with the outside environment.

To restructure this code, such that it will work for various thermocouple positions and a right-side convection condition, slight modifications to the base code was necessary. This consists of adding the index conversion from section 1.4 and changing the right-side boundary condition to include convection. These are relatively minor changes but drastically change the results.

The primary consideration is to minimize variation in the temperature distributions along the boundary. To help with visualization, the top and right boundary temperatures were extracted for each time interval and plotted. This creates two surface plots where the temperature variations are visible.

2.2 – Mean deviation Calculator for a Given Thermocouple Position

An evaluative mechanism is necessary to objectively compare the amount of deviation for a given thermocouple position. The code for this is found in Appendix A.3. For optimal results, there are two ends to meet:

1. Minimize fluctuations
2. Be as close to the reference temperature as possible

The mean deviation parameter can be used to consider both. This is very similar to evaluating the Ra value for a rough surface – just now it's being applied to the roughness of the temperature

profile over time concerning a reference temperature. The formula used is provided below in Equation 5.

$$mean\ absolute\ deviation = \frac{1}{N}\left(\sum |T - T_r|\right) \quad (5)$$

In this, T is the temperature across the relevant spatial domain for the entire time duration, T_r is the reference temperature, and N is the number of nodes within the relevant spatial and temporal domain. A smaller mean absolute deviation indicates better performance.

So, this code computes the linear system precisely the same as in section 2.1 with the additional consideration of the mean deviation. The primary output is the average mean deviation for the top and right boundaries. This function also outputs the mean deviation for just the top or right boundary, the average temperature and standard deviation along each boundary, and the time required to run the code. This code is designed to be used in an iterative solver, so the plotting and display functionalities from the previous code were removed in pursuit of faster solution times.

2.3 – Discrete Optimization for Mean Deviation Calculator

With the mean deviation function created, it can then be run within a progressive solver to determine which thermocouple position yields the optimal results. The code for this can be found in Appendix A.4.

This is immediately difficult because most numerical methods for 2D optimization consider a continuous function whose minimum converges to a specific point in space. This presents a problem since the model formulation is based on a series of discrete integer indexes that cannot be further subdivided. Therefore, the optimization must exclusively consider an array of indexes without being able to converge on non-integer values.

The computation times complicate this further. For a single index, the average computation time for the mean deviation calculator was between 4.9 and 5.1 seconds. This is not an issue for single computations, but for optimizing a domain of 1275 indices, the solution time becomes approximately 105 minutes. An optimization method with a minimal amount of calculations is thusly required.

After considering various numerical methods, a methodology was designed that scans the domain and picks the local area of lowest mean deviation to investigate further. Ultimately, this method cuts the calculation time down to about 12 minutes. Though there are likely better optimization algorithms for this application, the optimizer developed works sufficiently well while being simple.

To scan the domain, the matrix discretization was subdivided into a larger grid of 3×3 squares. The position of the center of each square is used to compute the mean deviation for that point. The code then searches for the index associated with the smallest mean deviation. When found, all positions within the 3×3 square are input into the mean deviation function. This process cuts the number of node calculations from 1275 to 144 – an 86% reduction.

A visualization of how this algorithm works is presented below in Figure 3.

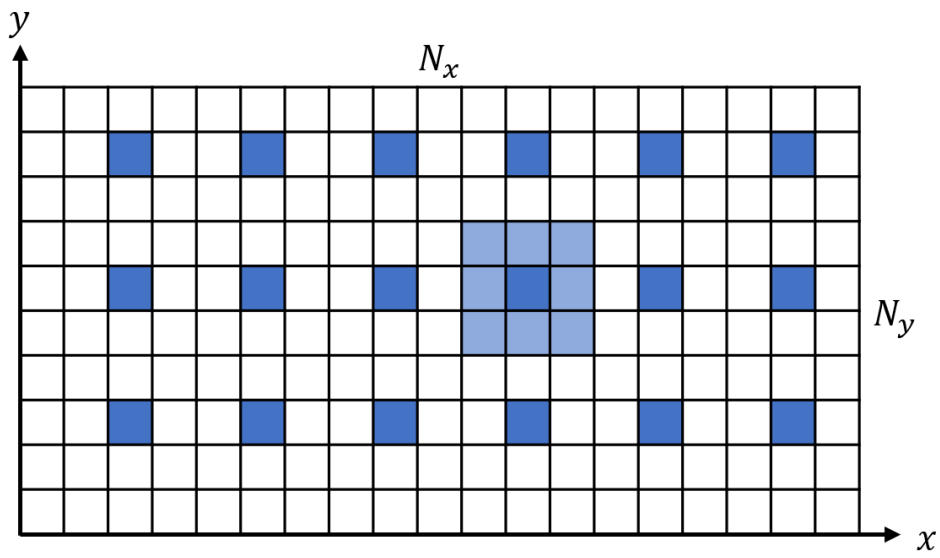


Figure 3. Visualization of Discretized Optimization Methodology

There are several issues with this method. Foremost, it omits the points that do not fit precisely within the 3×3 square grid. For example, in Figure 3, the leftmost column and the bottom row would not be considered. This method also assumes a continuous 2D function that can be approximated by sampling a ninth of the total points. If the mean deviation function unexpectedly increases or decreases, this optimization method is unlikely to appropriately consider it.

If these assumptions and risks are assumed, the optimization method works well. Thankfully, the issues anticipated are not presented in the results.

2.4 – Data Analysis

Another point of interest in the analysis is the effect of the heat generation gain constant k_g on the system. The previous code, the discrete optimization method, can only compute the mean deviation along the domain for a specific k_g value. So that code must be run multiple times with the constant changed manually. This was done a total of 5 times for k_g values of 1.4×10^4, 1.6×10^4, 1.8×10^4, 2.0×10^4, and 2.2×10^4.

This is a lot of data to process individually after running the discrete optimization method – instead, the workspaces in MATLAB were saved to be analyzed within another script. Hence, a new script was made exclusively to extract the relevant data and visualize the results. This script, found in Appendix A.5, generates four items for each k_g value sampled:

1. Evaluate the mean deviation at various points
2. Determine where the minimum mean deviation is
3. Plot the mean deviation surface as a function of thermocouple position
    a. For the average mean deviation
    b. For the top boundary mean deviation
    c. For the right boundary mean deviation
4. Plot the mean deviation of the various points as a function of k_g

16 figures are generated from this which each describes the effect of the controls system in various capacities.

3 – Model Verification

Each code utilizes the same code described in section 2.1. To verify all the code, all that needs to be done is to verify the first one. A slight modification to the boundary condition must be made such that the right edge no longer undergoes convection. With this update, and running the temperature distribution code at the point x = 0.02 m, y = 0.01 m, the following result is attainted.

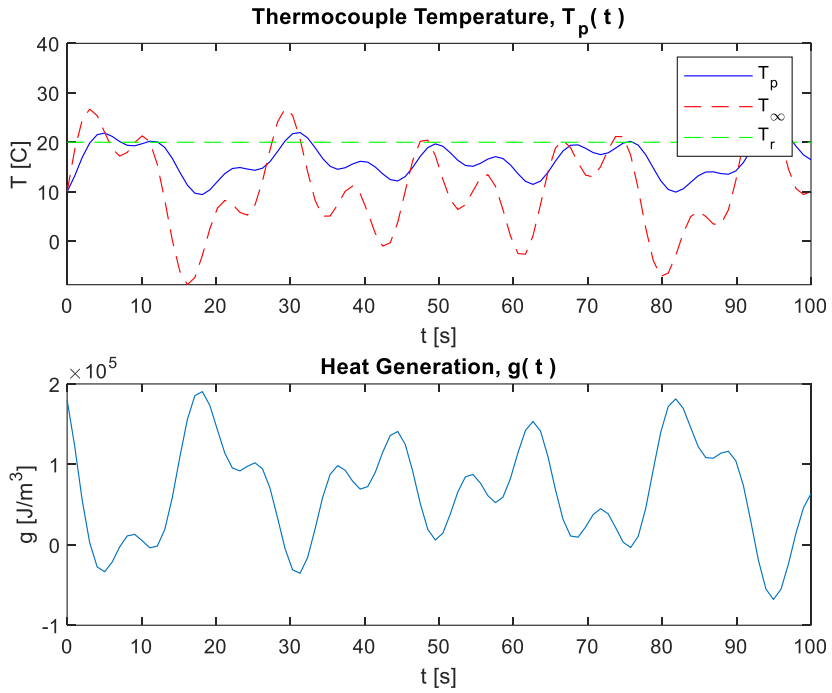


Figure 4. Thermocouple Temperature and Heat Generation in Developed Code (P = (0.02, 0.01) m; $k_g = 1.8\times10^4$)

Running Professor Trelles' code under these same conditions, Figure 5 is obtained.

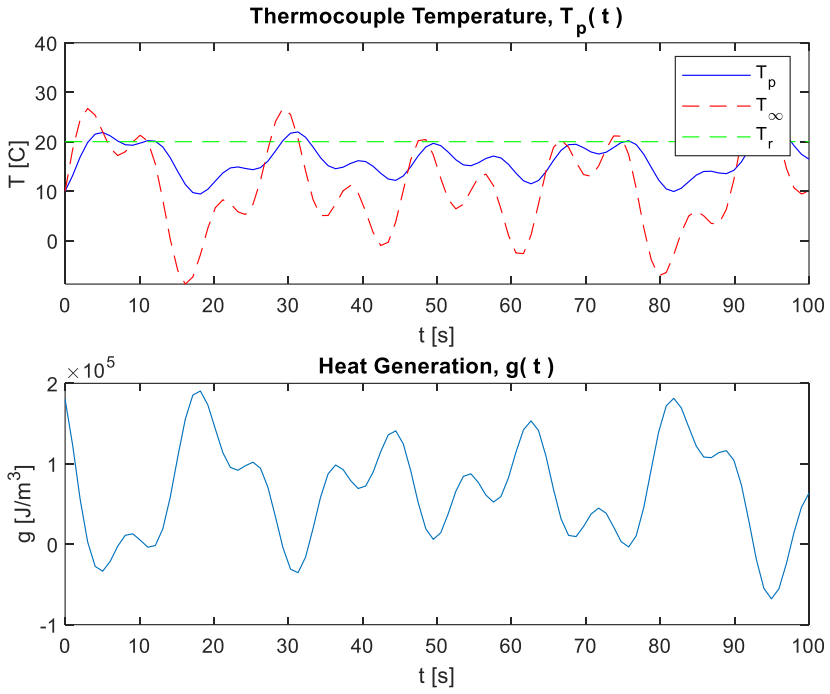


Figure 5. Known System Response (P = (0.02, 0.01) m; $k_g = 1.8\times10^4$)

Figures 4 and 5 are identical. When investigating these plots in MATLAB, each minima and maxima have the same magnitudes and positions. The outputs from each code yield the same result which implies the developed codes are accurate.

In brief, the reference and developed code produce the same result thereby verifying the developed code.

4 – Device Performance

4.1 – Boundary Plots

Now, with the code verified, an investigation must be conducted as to which thermocouple point within the spatial domain corresponds with the best performance.

Sample temperature distributions of the top and right boundaries are provided below in Figure 6 for a thermocouple at P = (0.01, 0.005) m and $k_g = 1.8\times10^4$.

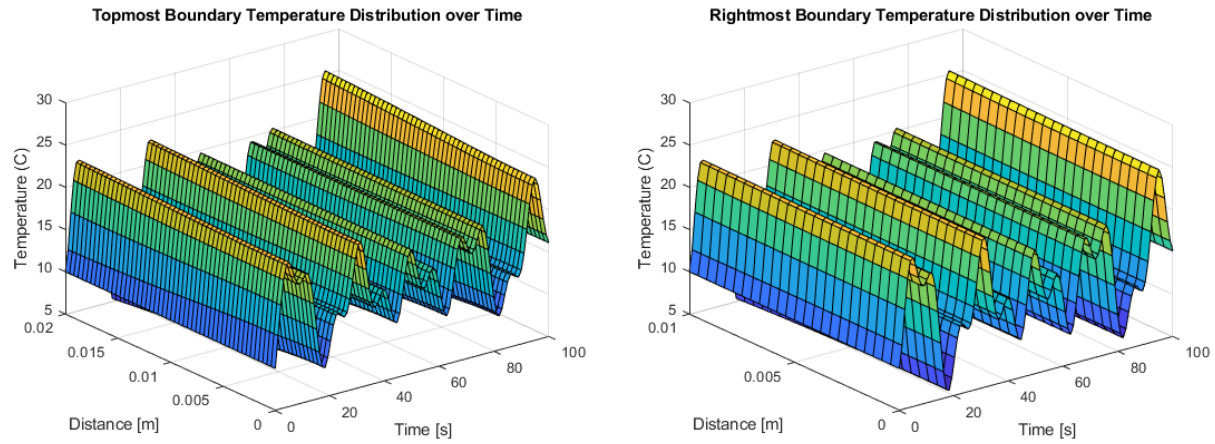


Figure 6. Top and Right Boundary Temperature Distributions over Time (P = (0.01, 0.005) m; $k_g = 1.8\times10^4$)

This type of behavior is consistent for most thermocouple points. For example, at P = (0.005, 0.005) m and $k_g = 1.8\times10^4$, the results are shown below in Figure 7.

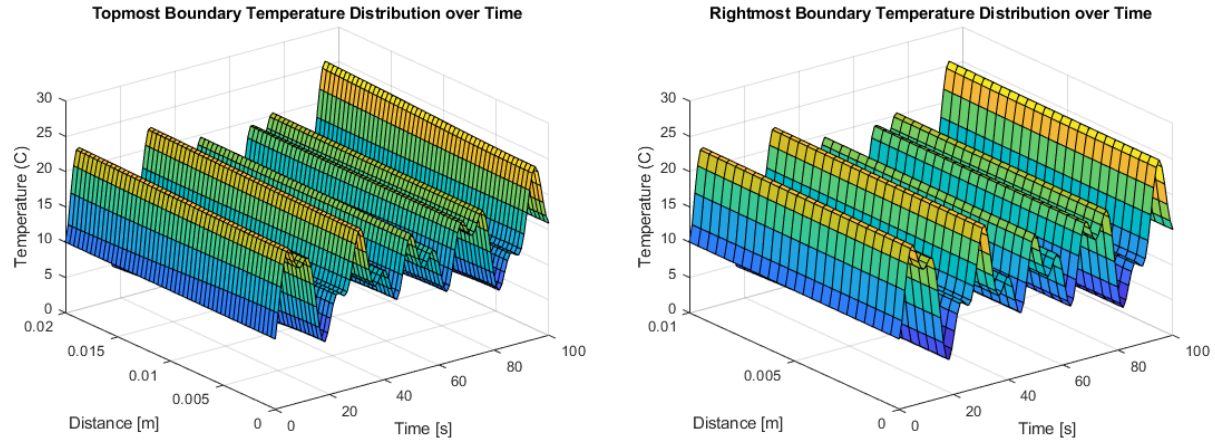


Figure 7. Top and Right Boundary Temperature Distributions over Time (P = (0.005, 0.005) m; $k_g = 1.8\times10^4$)

The differences between the plots in Figures 5 and 6 are barely perceptible. This further justifies using the mean absolute deviation measurement as a tool to compare different thermocouple positions – visual comparison is unacceptable.

4.2 – Optimization Results

The six points shown below in Figure 8 were sampled for several k_g values.

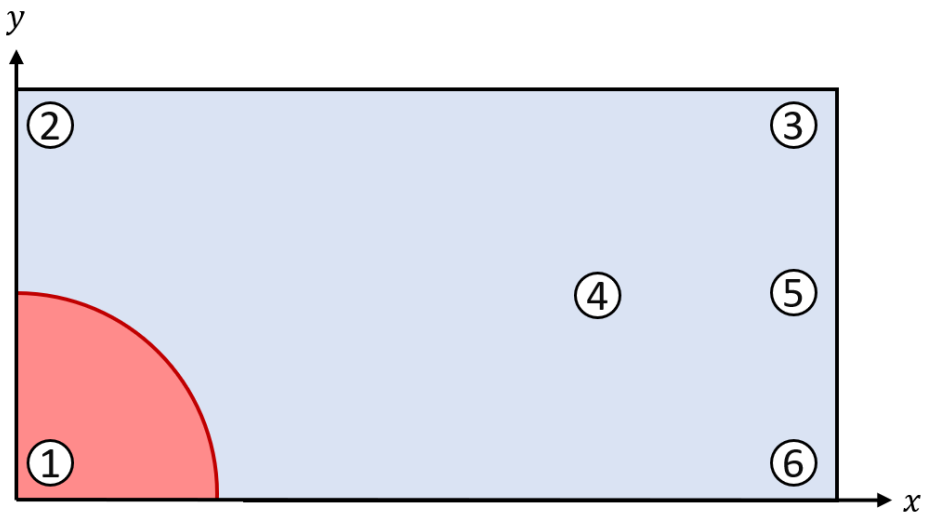


Figure 8. Points Sampled on the Controlled Surface
(1:(0, 0); 2:(0, y_{max}); 3:(x_{max}, y_{max}); 4:(¾ x_{max}, ½ y_{max}); 5:(x_{max}, ½ y_{max}), 6:(x_{max}, 0))

The mean absolute deviations for these six points are provided below in Table 1. Note, this mean absolute deviation is the average mean absolute deviation for the top and right boundaries

Table 1. Mean Absolute Temperature Deviations for Various k_g Values [°C]

Point	**k_g =1.4e4**	**k_g = 1.6e4**	**k_g =1.8e5**	**k_g = 2.0e5**	**k_g =2.2e6**
1	6.183	5.958	5.761	5.588	5.434
2	5.880	5.613	5.378	5.169	4.981
3	5.410	5.072	4.770	4.497	4.249
4	5.558	5.242	4.961	4.708	4.479
5	6.006	5.757	5.538	5.344	5.171
6	5.506	5.183	4.893	4.633	4.398

This data is also plotted below in Figure 9.

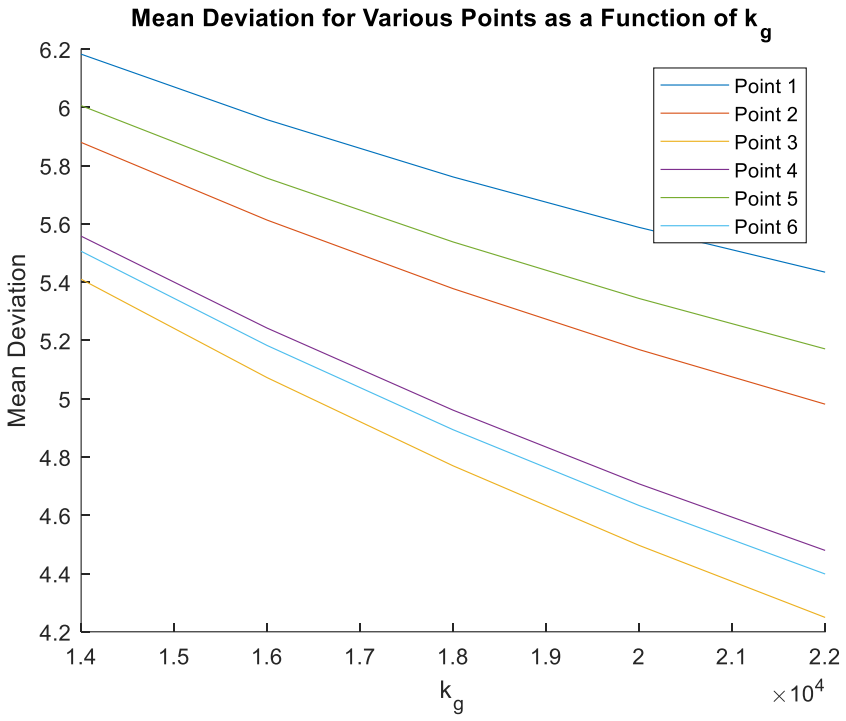


Figure 9. Mean Absolute Temperature Deviation as a Function of k_g

Clearly, from this plot, the thermocouple position that offers the most control is point 3 in Figure 8. This corresponds to a $d_x = 0$ and $d_y = 0$ from Figure 1. This means the new design should not move the thermocouple – it is already placed at a location that will minimize the temperature deviation. Furthermore, it indicates that performance improves as k_g increases.

4.3 – Further Optimization and Considerations

To get a clearer picture of the behavior of the surface, additional parameters beyond the mean distribution were considered including the mean temperature and sample standard deviation along the top and right boundaries. Each of these parameters was considered across the entire 2D domain. The plots for these, at $k_g = 2.2\times10^4$, are provided below in Figures 10 – 11. This value of k_g was selected since it corresponds with the best performance out of the sampled points.

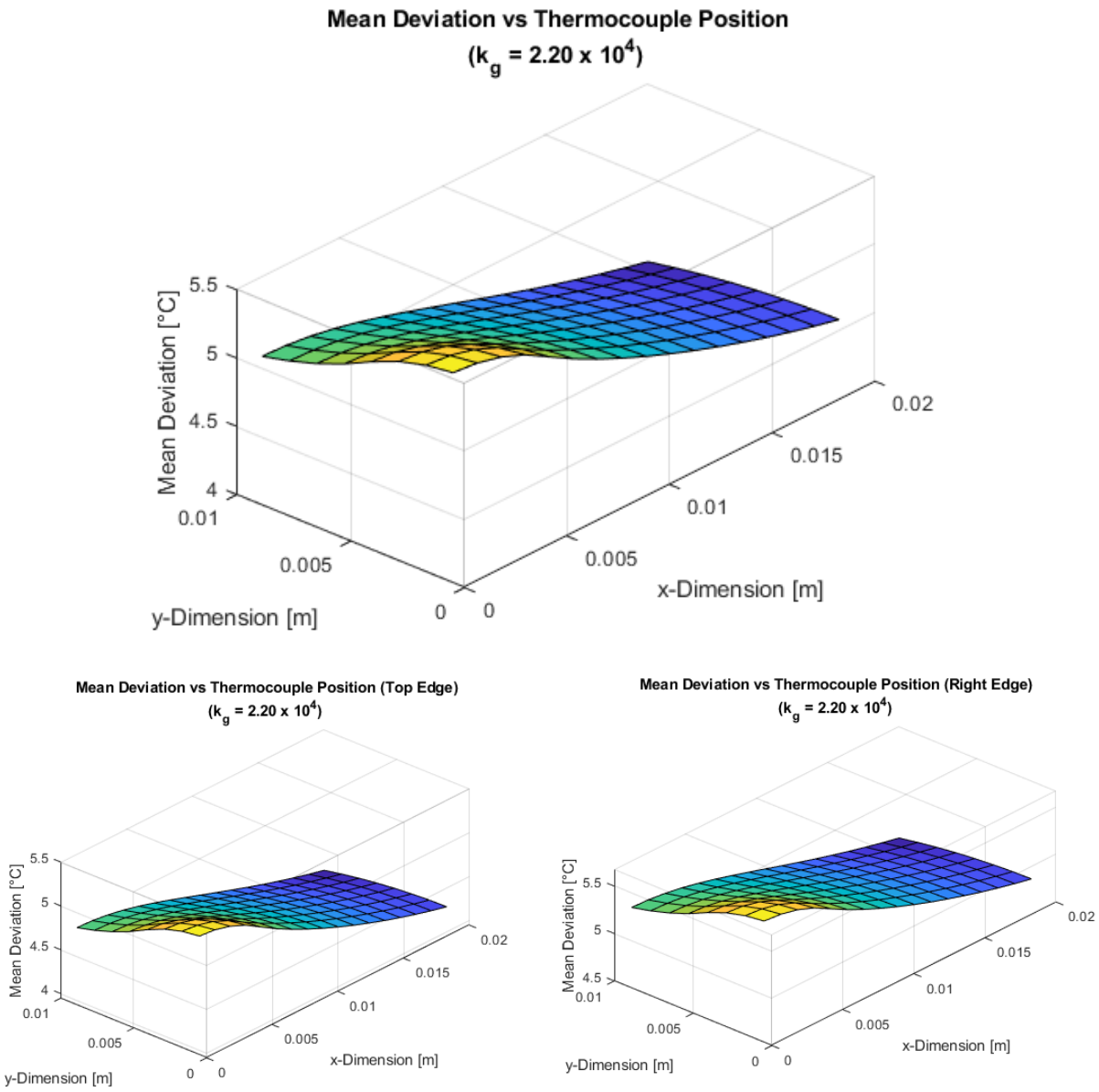


Figure 10. Surface Plots of Average Mean Deviation and Local Mean Deviation as a function of Thermocouple Position for kg = 2.2×10^4

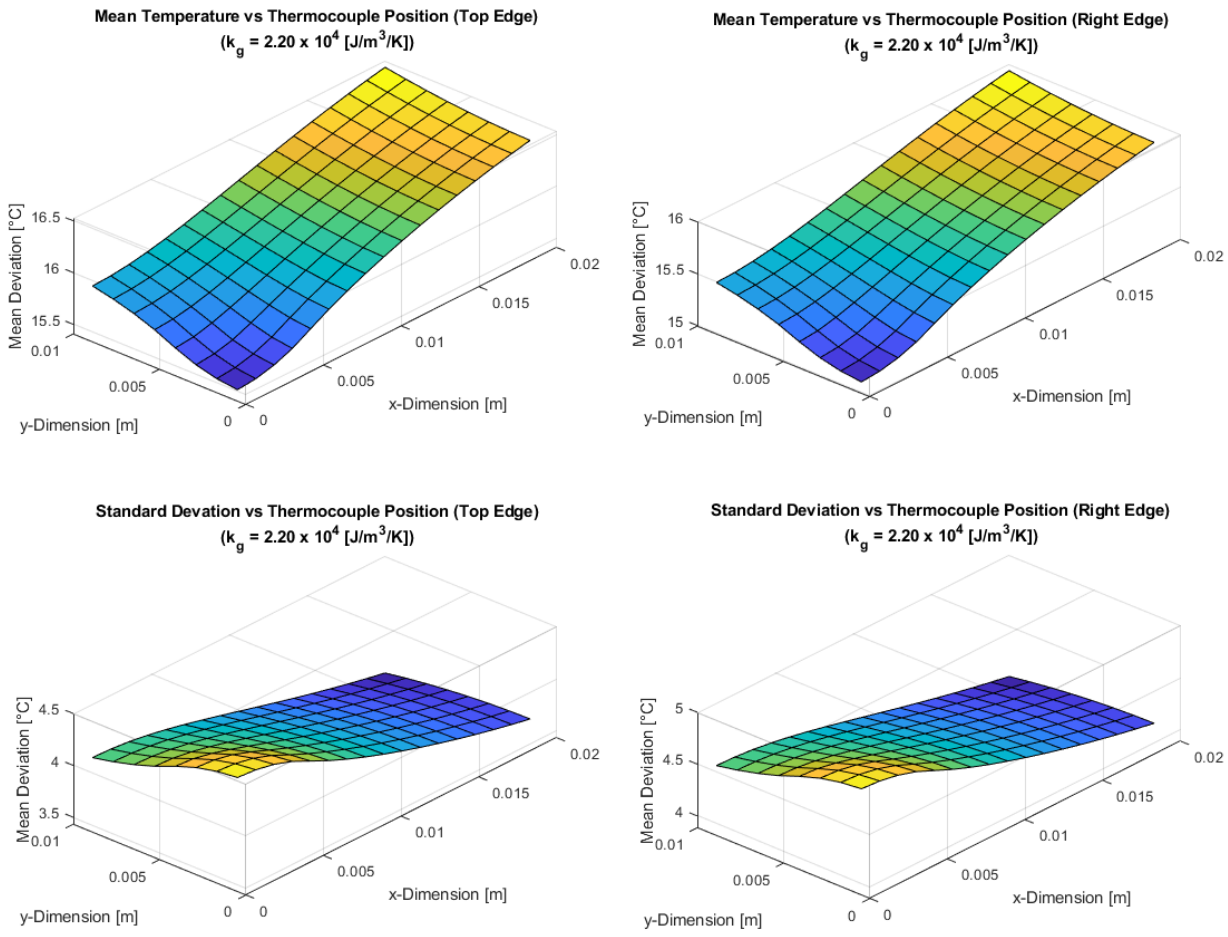


Figure 11. Average Temperature, and Sample Standard Deviation for the Top and Right Boundaries as a function of Thermocouple Position for kg = 2.2×10^4

These 7 plots create a much clearer image of how this system behaves. In each case, the performance improves as the thermocouple position approaches the upper right corner (P = (0.02, 0.01) m). This means that the mean deviations and standard deviations decrease as the average temperature increases towards the desired reference temperature. This is fantastic – it means the two evaluative conditions described in section 2.2 (lower variability and closer average temperature) are both maximally satisfied. There is no possible position other than P = (0.02, 0.01) m that will better satisfy these imposed conditions. Furthermore, for every k_g value tested, the mean deviation follows the same behavior seen in Figure 10 (these plots are provided in Appendix A.1).

Additionally, the code from section 2.5 used to process the data, the smallest average mean variation was searched for and a value of x = 0.02000 m and $\underline{y}$ = 0.01000 m was the output for every k_g tested. So, invariably, the optimal thermocouple position is at P = (0.02, 0.01) m.

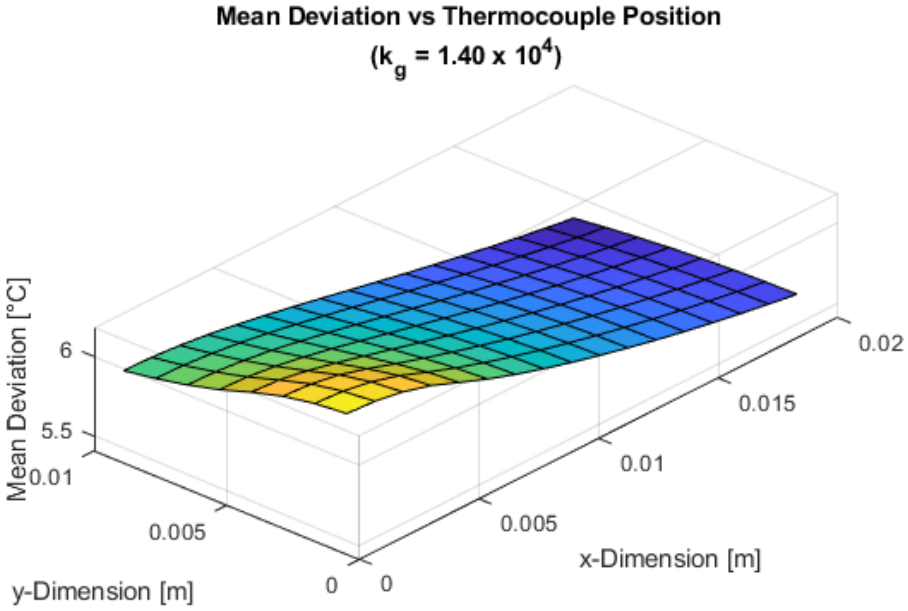
Bibliography

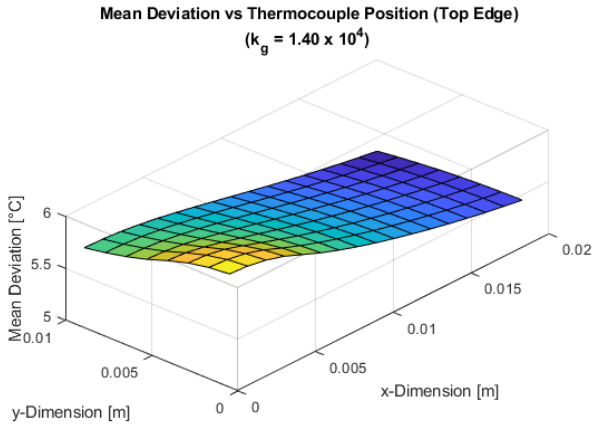
[1] J. P. Trelles, "MECH.5410 Advanced Heat Transfer; Project 3: Feedback temperature control," University of Massachusetts, Lowell, Lowell, 2020.

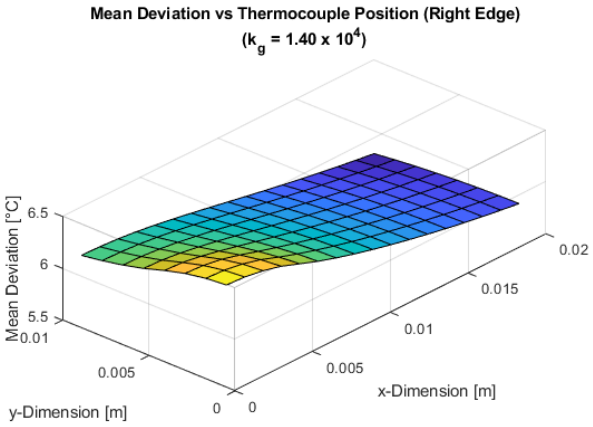
Appendix

A.1 – Characteristic Plots for All k_g Values Sampled

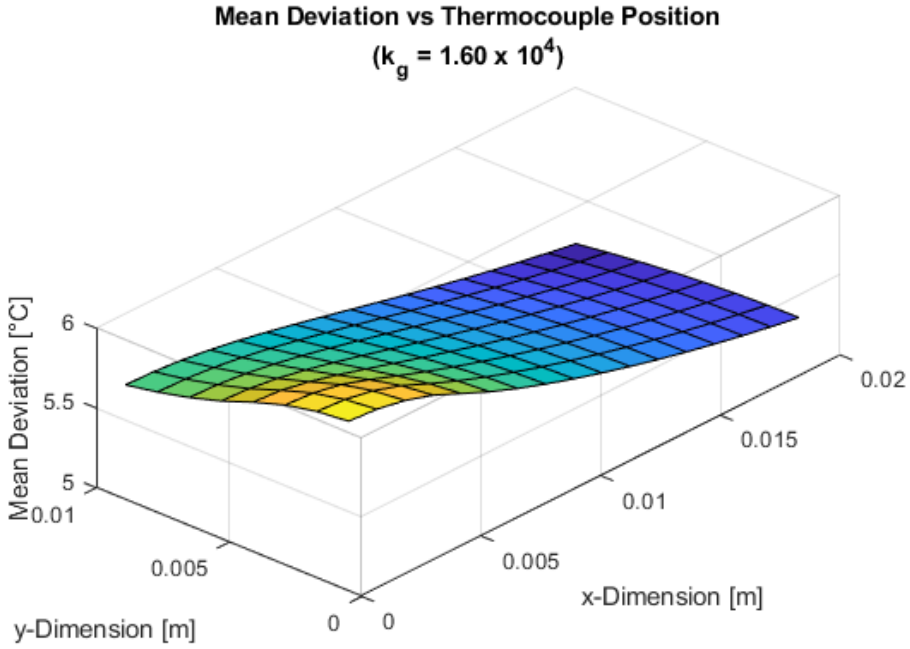
A.1.1 – $k_g = 1.4 \times 10^4$

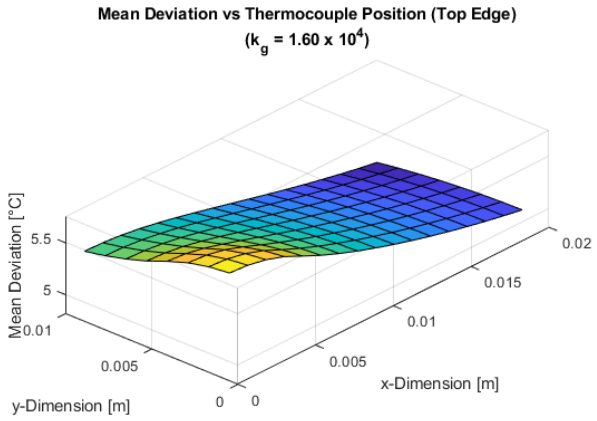


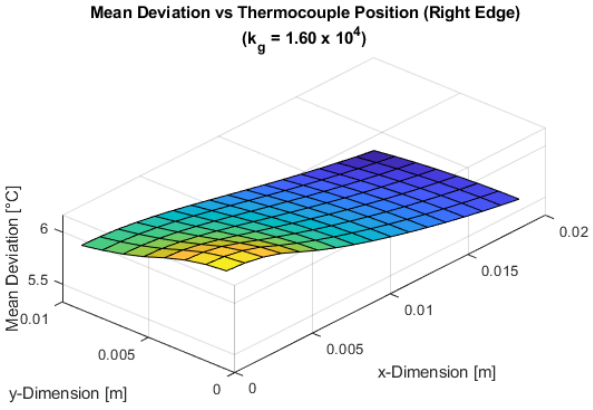




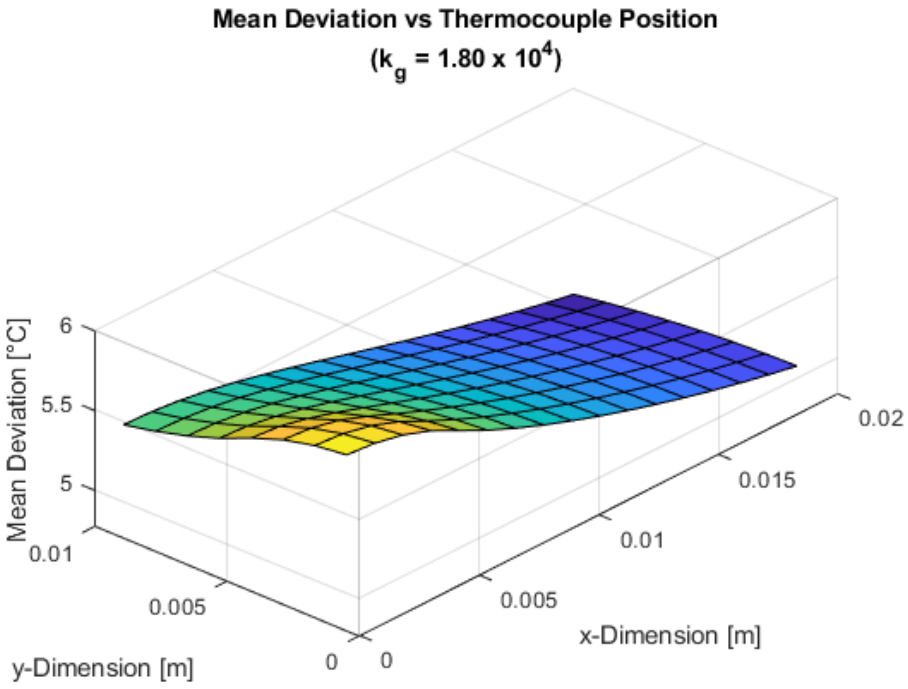
A.1.2 – $k_g = 1.4 \times 10^4$

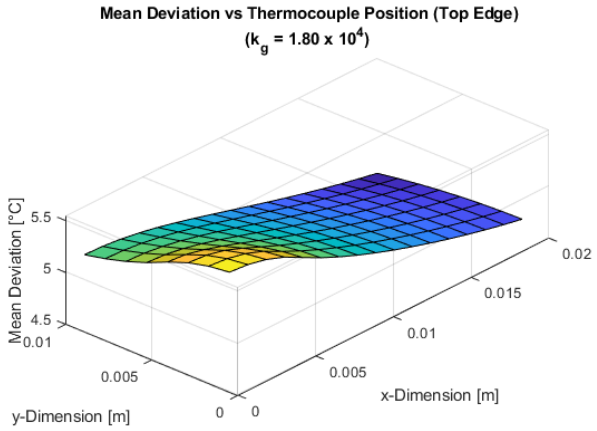


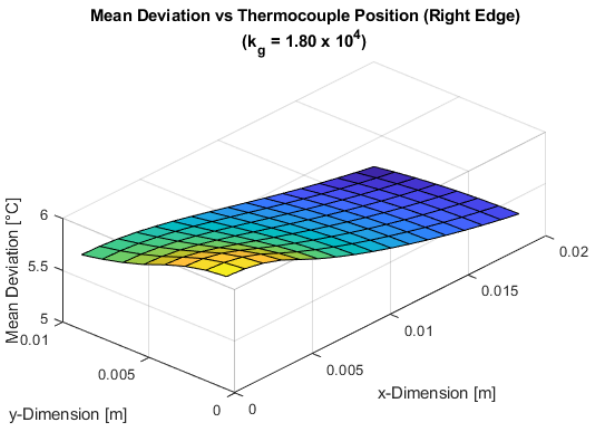




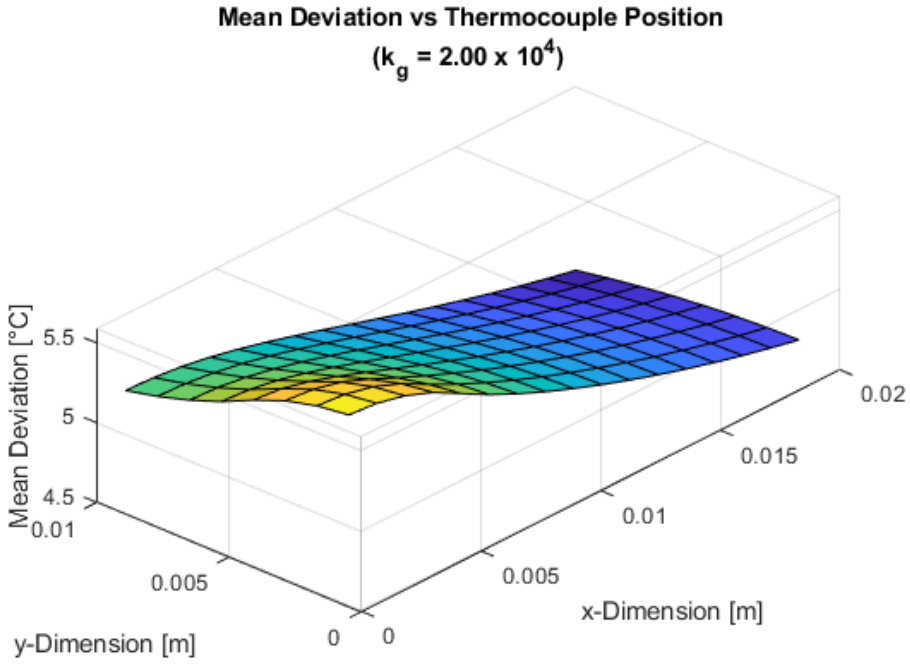
A.1.3 – $k_g = 1.8 \times 10^4$

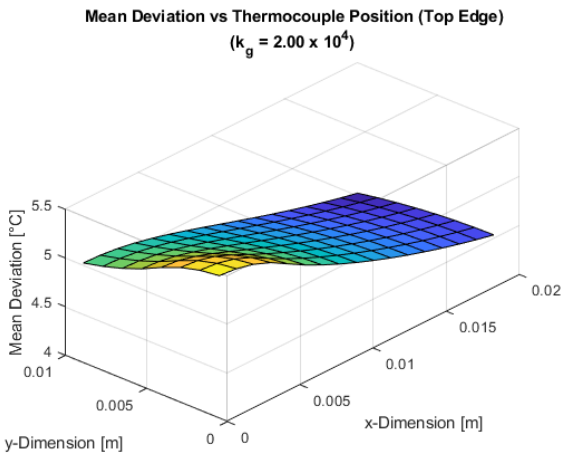


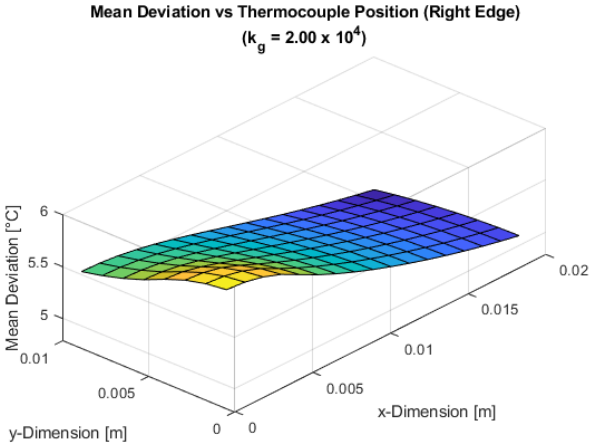




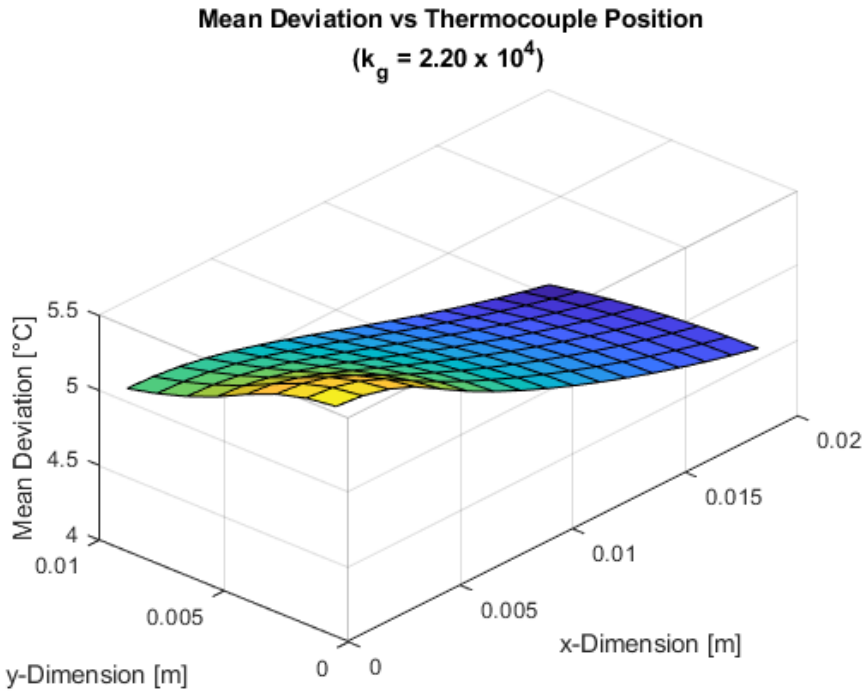
A.1.4 – $k_g = 2.0 \times 10^4$

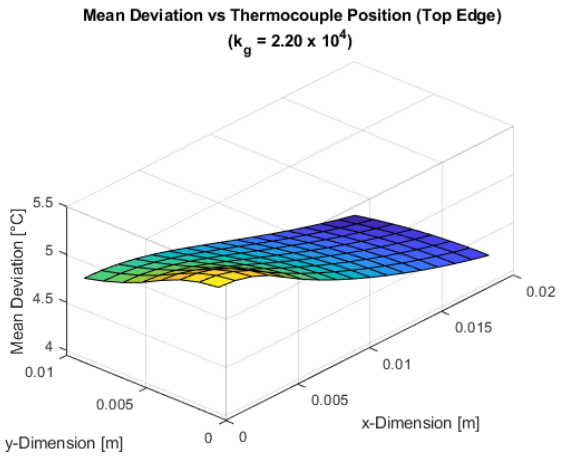


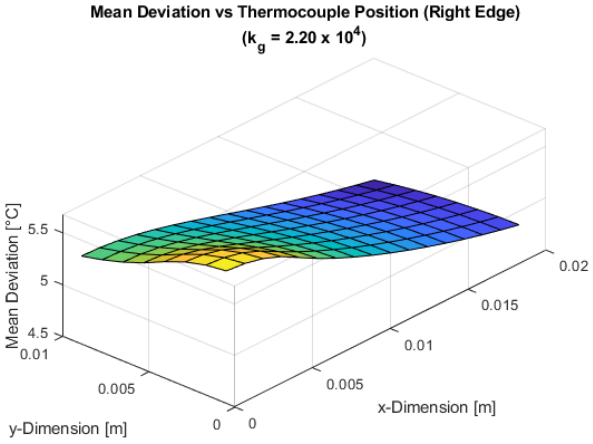




A.1.5 – $k_g = 2.2 \times 10^4$







A.2 – Temperature Distribution as a Function of Thermocouple Position

```
% Temperature Distribution as a Function of Thermocouple Position
%
% MECH.5410 - Advanced Heat Transfer
% Project 3
% Nicholas McLaughlin
% 11/9/2020

% NOTE ------------------------------------------------------------------
% This code is a modification of Professor Juan Pablo Trelles' 2020 code
% "heatplate_feedcontrol.m". Slight modifications have been made, such as
% adding the mean deviation functionality, but the core functionality is
% Professor Trelles' work.

% PURPOSE ---------------------------------------------------------------
% This code evaluates the temperature distributions for a desired
% thermocouple position.

% CODE DESCRIPTION ------------------------------------------------------
% This code is nearly identical to Professor Trelles' code. There are a
% few minor changes in order to consider different thermocouple positions
% and an update to the boundary condition. These are minor and the code
% operates nearly identically as "heatplate_feedcontrol.m".

% MODEL DESCRIPTION-----------------------------------------------------
% NOTE: This section is taken from Professor Trelles' code since it is
% nearly identical in functionality:
%
% Feedback control of a flat plate with a heating element.
%
% The temperature distribution T(x,y, t) through a rectangular domain of
% size Lx x Ly, with volumetric heat capacity rhoCp, thermal conductivity
% k, and volumetric heat generation g is described by:
%
%           rhoCo * dT/dt = k * d^2T/dx^2 + k * d^2T/dy^2 + g
%
% The term g( t ) is positive only within the conductor region - the
% control parameter, and zero elsewhere in the domain.
%
% The goal of the control approach is, given some (constant) reference
% temperature Tr and some varying Tinf(t), adjust the value of g such
% that the value of temperature at the point p (upper-right corner) Tp
% approaches Tr. The practical motivation of the problem is to maintain the
% top boundary (y = Ly) close to the reference temperature Tr.
%
% The initial condition for the problem is: ==============================
%
%             T = Tinitial, 0 < x < Lx, 0 < y < Ly, t = 0
%
% State-Space representation of the dynamic system:
%
%    dT/dt = A*T + B*u       ... (1)
%      y   = C*T             ... (2)
%      u   = K*(r - y)       ... (3)
%
% T: temperature of nodal values of temperature
```

```
% A: system matrix
% B: input matrix
% y: output vector,     y = [Tp; 0]
% C: output matrix,     C = [delta(ip, jp)], ip, jp: node number of p
% u: control vector,    u = [g; Tinf]
% K: gain matrix,       K = [kg 0; 0 1], (g = kg*(Tr - Tp))
% r: reference vector,  r = [Tr; Tinf]
%
% Solution:
%
% T^(n+1) = (I - dt*(A - B*K*C))\(T^n + dt*B*K*r )
%
% Finite Difference Method (FDM) approximation:
% Physical domain:
%        y
%        ^
%        |
% (0, Ly)|               (top)           (Lx, Ly)
%        o---------------------------------o P <-- location of thermocouple
%        |                                 |
%        |                                 |
%        |                                 |
%        |                                 |
%        |                                 |
%        |                                 |
%  (left)|____                             |(right)
%        |    \__   (radius R)             |
%        |       \_                        |
%        |         \_                      |
%        | conductor \                     |
%        |            |                    |
%        o---------------------------------o ----> x
%  (0, 0)             (bottom)          (Lx, 0)
%
% Index domain:
%
%  (1, 1)            (bottom)         (1, Nx)
%        o---------------------------o -----> j (equivalent to x axis)
%        |                            |
%        |                            |
%        |                            |
%        |           i-1,j            |
%        |             |              |
%  (left)|    i,j-1 - i,j - i,j+1     |(right)
%        |             |              |
%        |           i+1,j            |
%        |                            |
%        |                            |
%        |                            |
%        o---------------------------o
%  (Ny,1)|             (top)          (Ny, Nx)
%        |
%        i (equivalent to y axis)

% BOUNDARY CONDITIONS ----------------------------------------------------
% The boundary conditions are as follows for each edge:
%
```

```
% Bottom:      k * dT/dx = 0
% Left:       -k * dT/dx = 0
% Top:         k * dT/dy = h*(T -Tinf(t))
% Right:      -k * dT/dy = h*(T -Tinf(t))
%
% Essentially, this means there is no heat transfer at the bottom and left
% boundaries but there is convective heat transfer at the top and right
% boundaries. This is consistent with the requirements of the analysis.

% BEGINNING OF CODE ------------------------------------------------------
clear; close all

% Specify which nodes are of interest
Px = 13;
Py = 13;

% Control parameters
Tr       =   20;     % [C]       reference temperature (20)
kg       = 1.8e4;    % [J/m^3/K] control parameter (1.8e4)

% Spatial domain
Lx       = 0.020;    % [m]       length of domain along x
Ly       = 0.010;    % [m]       length of domain along y
R        = 0.005;    % [m]       radius
Nx       =   51;     % number of nodes along x (101, 51)
Ny       =   25;     % number of nodes along y (50, 25)

% Temporal domain
tfinal   = 1e2;      % [s]       final time (1e2)
Nt       =  100;     % number of temporal nodes (100)

% Properties:
rhoCp    = 2e3;      % [J/m^3/K] volumetric heat capacity
k        = 1.0;      % [W/m/K]   thermal conductivity
h        = 10.0;     % [W/m^2/K] convective coefficient

% Initial conditions
Tinitial =   10;     % [C]       initial temperature

% Function describing the time dependency of external temperature
funTinf = @( t )( 10 + 10 * sin( 9 * pi * t / tfinal ) + ...
    7 * sin(22 * pi * t / tfinal ) + ...
    5 * sin( 7 * pi * t / tfinal ) );

% Convert specified matrix position to an index
Pnode = Nx * ( Py - 1 ) + Px;
Ctx.node = Pnode;

% Store all constants
Ctx.rhoCp = rhoCp;
Ctx.k     = k;
Ctx.h     = h ;
Ctx.Tr    = Tr;
Ctx.kg    = kg;
Ctx.Lx    = Lx;
Ctx.Ly    = Ly;
Ctx.R     = R ;
```

```
Ctx.Nx    = Nx;
Ctx.Ny    = Ny;

% Calculate solver parameters & properties
N = Nx * Ny;               % total number of unknowns (nodes)

dx = Lx / ( Nx - 1 );      % discrete x differential
x  = 0 : dx : Lx;          % discrete x axis, x(1) = 0, x(Nx) = Lx

dy = Ly / ( Ny - 1 );      % discrete y differential
y  = 0 : dy : Ly;          % discrete y axis, y(1) = 0, y(Ny) = Ly

dt = tfinal / ( Nt - 1 );  % discrete t differential
t  = 0 : dt : tfinal;      % discrete t axis, t(1) = 0, t(Nt) = tfinal
tcoord = t;

% Preallocate space and define identity matrix
T = zeros( N, Nt );        % solution for each time interval
I = speye( N, N  );        % identity matrix

% Calculate the system matrices for the given constants
[A, B, C, K] = funSys(Ctx);

% Populate T with the initial conditions
T(:, 1) = Tinitial;

% SOLUTION ----------------------------------------------------------------
for n = 1 : Nt-1

    % Populate element of linear system
    r = [ Tr; funTinf( t( n ) ) ];

    % Solve linear system
    T( :, n + 1 ) = ( I - dt * ( A - B * K * C ) ) \ ...
        ( T( :, n ) + dt * ( B * K * r ) );

end

% COLLECT DATA FROM SPECIFIC TIMES ----------------------------------------
% Set of instants as percentage of total time
nt1 = 2;                   % ~ 0%, initial
nt2 = round( 0.20 * Nt );  %  20%
nt3 = round( 0.50 * Nt );  %  50%
nt4 = Nt;                  % 100%, final

% Create x-y grid
[ Xgrid, Ygrid ] = meshgrid( x, y );

% Arrange solution into x-y grid
Tgrid1 = zeros( Ny, Nx );
Tgrid2 = zeros( Ny, Nx );
Tgrid3 = zeros( Ny, Nx );
Tgrid4 = zeros( Ny, Nx );

for i = 1 : Ny
    for j = 1 : Nx
        node = Nx * ( i - 1 ) + j;
```

```
        Tgrid1( i, j ) = T( node, nt1 );
        Tgrid2( i, j ) = T( node, nt2 );
        Tgrid3( i, j ) = T( node, nt3 );
        Tgrid4( i, j ) = T( node, nt4 );
    end
end

% PLOTTING ------------------------------------------------------------------
% Surface plots
figure
subplot( 2, 2, 1 )
surfc( Xgrid, Ygrid, Tgrid1 )
view( 2 )
axis image
colorbar
xlabel(' x [m] ')
ylabel(' y [m] ')
zlabel(' T [C] ')
title(' T( x, y, ~t_{initial}) ')

subplot( 2, 2, 2 )
surfc( Xgrid, Ygrid, Tgrid2 )
view( 2 )
axis image
colorbar
xlabel(' x [m] ')
ylabel(' y [m] ')
zlabel(' T [C] ')
title(' T( x, y, 0.2t_{final} ) ')

subplot( 2, 2, 3 )
surfc( Xgrid, Ygrid, Tgrid3 )
view( 2 )
axis image
colorbar
xlabel(' x [m] ');
ylabel(' y [m] ');
zlabel(' T [C] ')
title(' T( x, y, 0.5t_{final} ) ')

subplot( 2, 2, 4 )
surfc( Xgrid, Ygrid, Tgrid4 )
view( 2 )
axis image
colorbar
xlabel(' x [m] ')
ylabel(' y [m] ')
zlabel(' T [C] ')
title(' T( x, y, t_{final} ) ')

% Temporal plots
figure

subplot( 2, 1, 1 )
plot( t, T( N, : ), '-b', ...
    t, funTinf( t ), '--r', ...
    t, Tr * ones( 1, Nt ), '--g' )
```

```
legend('T_p', 'T_{\infty}', 'T_r' )
xlabel(' t [s] ')
ylabel(' T [C] ')
title(' Thermocouple Temperature, T_p( t ) ')

subplot( 2, 1, 2 )
plot( t, kg * ( Tr - T( N, : ) ) )
xlabel(' t [s] ')
ylabel(' g [J/m^3] ')
title(' Heat Generation, g( t ) ')

% EXPORT BOUNDARY DATA ---------------------------------------------------

% First, lets consider the topmost boundary (the orignal convection
% condition)
Ttop = zeros(Ny,tfinal);
for t = 1:tfinal
    for i = Ny
        for  j = 1:Nx
            nodes = Nx * ( i - 1 ) + j;
            Ttop(j,t) = T(nodes,t);
        end
    end
end

% Second, lets consider the rightmost boundary (the second convection
% condition)
Tright = zeros(Ny,tfinal);
for t = 1:tfinal
    for i = 1:Ny
        for  j = Nx
            nodes = Nx * ( i - 1 ) + j;
            Tright(i,t) = T(nodes,t);
        end
    end
end

% Plot the temperature distributions at the boundaries as a function of
% time
figure
surf(tcoord,x,Ttop)
title('Topmost Boundary Temperature Distribution over Time')
xlabel('Time [s]')
ylabel('Distance [m]')
zlabel('Temperature (C)')

figure
surf(tcoord,y,Tright)
title('Rightmost Boundary Temperature Distribution over Time')
xlabel('Time [s]')
ylabel('Distance [m]')
zlabel('Temperature (C)')
toc

% NESTED FUNCTION DEFINING MATRICES --------------------------------------
function [A, B, C, K] = funSys(Ctx)
```

```
% RETRIEVE CONSTANTS ------------------------------------------------------
rhoCp    = Ctx.rhoCp;
k        = Ctx.k;
h        = Ctx.h ;
kg       = Ctx.kg;
Lx       = Ctx.Lx;
Ly       = Ctx.Ly;
R        = Ctx.R ;
Nx       = Ctx.Nx;
Ny       = Ctx.Ny;
Nx       = Ctx.Nx;
Ny       = Ctx.Ny;

% INITIALIZATIONS ----------------------------------------------------------
dx  = Lx / ( Nx - 1 );    % differential - x
dy  = Ly / ( Ny - 1 );    % differential - y
x   = 0 : dx : Lx;        % discrete x axis, x(1) = 0, x(Nx) = Lx
y   = 0 : dy : Ly;        % discrete y axis, y(1) = 0, y(Ny) = Ly
N   = Nx * Ny;            % total number of unknowns (nodes)
vln = 1.0e8;              % very large number
A = sparse( 1, 1, 1, N, N, 5 * N ); % sparse linear system matrix
B = sparse( N, 2 );                 % input matrix

% C MATRIX -----------------------------------------------------------------
C = sparse(2, N);        % output matrix
C(1, Ctx.node) =  1;     % Specify which node to consider

% GAIN MATRIX --------------------------------------------------------------
K = [ kg  0.0  ;         % gain matrix
    0.0 1.0 ];

% BOUNDARY CONDITIONS ------------------------------------------------------

% For the left boundary
j = 1;
for i = 1 : Ny
    node  = Nx*(i-1)+j;
    nodeb = node + 1;
    A(node, node)   = vln*(-k/dx);
    A(node, nodeb)  = vln*(k/dx);
end

% For the right boundary
j = Nx;
for i = 1 : Ny
    node  = Nx*(i-1)+j;
    nodeb = node - 1;

    % Can verify this with Professor Trelles' code by uncommeting this line
    % and commenting out the B(node, 2) line

    %A(node, node)   = vln*(-k/dx-h);
    A(node, node)   = vln*(-k/dx-h);
    A(node, nodeb)  = vln*(k/dx);
    B(node, 2)      = vln*(h);
end
```

```
% For the bottom boundary
i = 1;
for j = 2 : Nx-1
    node  = Nx*(i-1)+j;
    nodeb = node+Nx;
    A( node,node)   = vln*(-k/dy);
    A( node,nodeb)  = vln*(k/dy);
end

% For the top boundary
i = Ny;
for j = 2 : Nx-1
    node  = Nx*(i-1)+j;
    nodeb = node - Nx;
    A( node, node)  = vln*(-k/dy-h);
    A( node, nodeb) = vln*(k/dy);
    B( node, 2)     = vln*(h);
end

% Considering internal nodes
Cleft   =  k / rhoCp / dx^2;
Cright  =  k / rhoCp / dx^2;
Cbottom =  k / rhoCp / dy^2;
Ctop    =  k / rhoCp / dy^2;
Ccenter = -k / rhoCp  * ( 2.0 / dx^2 + 2.0 / dy^2 );

for i = 2 : Ny-1
    for j = 2 : Nx-1

        % current node:
        node = Nx * ( i - 1 ) + j;

        % elements of B:
        if sqrt( x( j )^2 + y( i )^2 ) <= R
            B( node, 1 ) = 1.0 / rhoCp;
        end

        % elements of A:
        node_left   = node -  1; % ( i    , j - 1 )
        node_right  = node +  1; % ( i    , j + 1 )
        node_bottom = node - Nx; % ( i - 1, j     )
        node_top    = node + Nx; % ( i + 1, j     )
        node_center = node     ; % ( i    , j     )

        A( node, node_left   ) = Cleft  ;
        A( node, node_right  ) = Cright ;
        A( node, node_bottom ) = Cbottom;
        A( node, node_top    ) = Ctop   ;
        A( node, node_center ) = Ccenter;

    end
end
end
```

A.3 – Mean Deviation Calculator for a Given Thermocouple Position

```
% Mean Deviation Calculator for a Given Thermocouple Position
%
% MECH.5410 - Advanced Heat Transfer
% Project 3
% Nicholas McLaughlin
% 11/9/2020

% NOTE ------------------------------------------------------------------
% This code is a modification of Professor Juan Pablo Trelles' 2020 code
% "heatplate_feedcontrol.m". Slight modifications have been made, such as
% adding the mean deviation functionality, but the core functionality is
% Professor Trelles' work.

% PURPOSE ---------------------------------------------------------------
% This code evaluates the statistical variability of the temperature with
% respect to a prescribed reference temperature. This is with respect to
% the problem description in Project 3. This is measured using the mean
% deviation (MD)

% CODE DESCRIPTION ------------------------------------------------------
% This particular code defines as the callable function Mean_Deviation in the
% following structure:
%
% [zavg,ztop,zright,avgtop,avgright,sigtop,sigright,TimeEnd] = ...
%                                     Mean_Deviation(Px,Py,Lx,Ly,Nx,Ny,kg)

% INPUTS ----------------------------------------------------------------
% Px: Node position in the x direction
% Py: Node position in the y direction
% Lx: Total length in the x direction
% Ly: Total length in the y direction
% Nx: Number of nodes in the x direction
% Ny: Number of nodes in the y direction
% kg: Heat generation gain constant

% OUTPUTS ---------------------------------------------------------------
% zavg      : the average MD of the top and right boundaries for the
%           time interval
% ztop      : the MD of the top boundary over the time interval
% zright    : the MD of the right boundary over the time interval
% avgtop    : the average temperature of the top boundary over the time
%           interval
% avgright  : the average temperature of the right boundary over the time
%           interval
% sigtop    : the standard deviation of the temperature at the top boundary
% sigright  : the standard deviation of the temperature at the right boundary
% TimeEnd   : the time it took for the code to finish

% MODEL DESCRIPTION-----------------------------------------------------
% NOTE: This section is taken from Professor Trelles' code since it is
% nearly identical in functionality:
%
% Feedback control of a flat plate with a heating element.
%
```

```
% The temperature distribution T(x,y, t) through a rectangular domain of
% size Lx x Ly, with volumetric heat capacity rhoCp, thermal conductivity
% k, and volumetric heat generation g is described by:
%
%           rhoCo * dT/dt = k * d^2T/dx^2 + k * d^2T/dy^2 + g
%
% The term g( t ) is positive only within the conductor region - the
% control parameter, and zero elsewhere in the domain.
%
% The goal of the control approach is, given some (constant) reference
% temperature Tr and some varying Tinf(t), adjust the value of g such
% that the value of temperature at the point p (upper-right corder) Tp
% aproaches Tr. The practical motivation of the problem is to maintain the
% top boundary (y = Ly) close to the reference temperature Tr.
%
% The initial condition for the problem is: ==============================
%
%             T = Tinitial, 0 < x < Lx, 0 < y < Ly, t = 0
%
% State-Space representation of the dynamic system:
%
%    dT/dt = A*T + B*u       ... (1)
%      y   = C*T             ... (2)
%      u   = K*(r - y)       ... (3)
%
% T: temperature of nodal values of temperature
% A: system matrix
% B: input matrix
% y: output vector,    y = [Tp; 0]
% C: output matrix,    C = [delta(ip, jp)], ip, jp: node number of p
% u: control vector,   u = [g; Tinf]
% K: gain matrix,      K = [kg 0; 0 1], (g = kg*(Tr - Tp))
% r: reference vector, r = [Tr; Tinf]
%
% Solution:
%
% T^(n+1) = (I - dt*(A - B*K*C))\(T^n + dt*B*K*r )
%
% Finite Difference Method (FDM) approximation:
% Physical domain:
%        y
%        ^
%        |
% (0, Ly)|               (top)           (Lx, Ly)
%        o---------------------------------o P <-- location of thermocouple
%        |                                 |
%        |                                 |
%        |                                 |
%        |                                 |
%        |                                 |
%        |                                 |
%  (left)|____                             |(right)
%        |    \__   (radius R)             |
%        |       \_                        |
%        |         \_                      |
%        | conductor \                     |
%        |            |                    |
```

```
%        o---------------------------------o ----> x
%  (0, 0)              (bottom)          (Lx, 0)
%
% Index domain:
%
%  (1, 1)            (bottom)         (1, Nx)
%        o----------------------------o -----> j (equivalent to x axis)
%        |                             |
%        |                             |
%        |                             |
%        |            i-1,j            |
%        |              |              |
%  (left)|    i,j-1 - i,j - i,j+1      |(right)
%        |              |              |
%        |            i+1,j            |
%        |                             |
%        |                             |
%        |                             |
%        o----------------------------o
%  (Ny,1)|             (top)          (Ny, Nx)
%        |
%        i (equivalent to y axis)

% BOUNDARY CONDITIONS ---------------------------------------------------------
% The boundary conditions are as follows for each edge:
%
% Bottom:      k * dT/dx = 0
% Left:       -k * dT/dx = 0
% Top:         k * dT/dy = h*(T -Tinf(t))
% Right:      -k * dT/dy = h*(T -Tinf(t))
%
% Essentially, this means there is no heat transfer at the bottom and left
% boundaries but there is convective heat transfer at the top and right
% boundaries. This is consistent with the requirements of the analysis.

% BEGINNING OF CODE -----------------------------------------------------------
function [Zavg,Ztop,Zright,avgtop,avgright,sigtop,sigright,TimeEnd] = ...
    Mean_Deviation(Px,Py,Lx,Ly,Nx,Ny,kg)

% Begin timer
TimeStart = tic;

% Control parameters:
Tr       =   20;     % [C]       reference temperature (20)

% Solver parameters
Nt       =  100;     % number of temporal nodes (100)
tfinal   = 1e2;      % [s]       final time (1e2)

% Properties:
R        = 0.005;    % [m]       radius
rhoCp    = 2e3;      % [J/m^3/K] volumetric heat capacity
k        = 1.0;      % [W/m/K]   thermal conductivity
h        = 10.0;     % [W/m^2/K] convective coefficient
```

```
% Initial condition:
Tinitial =   10;     % [C]       initial temperature

% Function describing the time dependency of external temperature
funTinf = @( t )( 10 + 10 * sin( 9 * pi * t / tfinal ) + ...
                   7 * sin(22 * pi * t / tfinal ) + ...
                   5 * sin( 7 * pi * t / tfinal ) );

% Convert specified matrix position to an index
Pnode = Nx * ( Py - 1 ) + Px;
Ctx.node = Pnode;

% Store all constants
Ctx.rhoCp = rhoCp;
Ctx.k     = k ;
Ctx.h     = h ;
Ctx.Tr    = Tr;
Ctx.kg    = kg;
Ctx.Lx    = Lx;
Ctx.Ly    = Ly;
Ctx.R     = R ;
Ctx.Nx    = Nx;
Ctx.Ny    = Ny;

% Calculate solver parameters & properties
N = Nx * Ny;               % total number of unknowns (nodes)

dx = Lx / ( Nx - 1 );      % discrete x differential
x  = 0 : dx : Lx;          % discrete x axis, x(1) = 0, x(Nx) = Lx

dy = Ly / ( Ny - 1 );      % discrete y differential
y  = 0 : dy : Ly;          % discrete y axis, y(1) = 0, y(Ny) = Ly

dt = tfinal / ( Nt - 1 );  % discrete t differential
t  = 0 : dt : tfinal;      % discrete t axis, t(1) = 0, t(Nt) = tfinal

% Preallocate space and define identity matrix
T = zeros( N, Nt );        % solution for each time interval
I = speye( N, N  );        % identity matrix

% Calculate the system matrices for the given constants
[A, B, C, K] = funSys(Ctx);

% Populate T with the initial conditions
T(:, 1) = Tinitial;

% SOLUTION ----------------------------------------------------------------
% Solution over the time interval
for n = 1:(Nt-1)

    % Populate elements of the linear system
    r = [ Tr; funTinf( t( n ) ) ];

    % Solve the linear system
```

```
    T(:, n+1) = (I - dt*(A - B*K*C))\(T(:, n) + dt*(B*K*r));
end

% COLLECT BOUNDARY DATA ---------------------------------------------------
% Preallocate space for Ttop and Tright
Ttop = zeros(Ny,tfinal);
Tright = zeros(Ny,tfinal);

% Consider the top boundary temperature in time
for t = 1:tfinal
    for i = Ny
        for  j = 1:Nx
            nodes = Nx * ( i - 1 ) + j;
            Ttop(j,t) = T(nodes,t);
        end
    end
end

% Perform statistical calculations on the top boundary data
sigtop = std2(Ttop);                % Standard deviation
avgtop = mean(Ttop,'all');          % Average temperature
[m,n] = size(Ttop);
N = m*n;
Ztop = (1/N)*(sum(abs(Ttop - Tr), 'all'));   % MD Top

% Consider the right boundary temperature in time
for t = 1:tfinal
    for i = 1:Ny
        for  j = Nx
            nodes = Nx * ( i - 1 ) + j;
            Tright(i,t) = T(nodes,t);
        end
    end
end

% Perform statistical calculations on the right boundary date
sigright = std2(Tright);            % Standard deviation
avgright = mean(Tright,'all');      % Average temperature
[m,n] = size(Tright);
N = m*n;
Zright = (1/N)*(sum(abs(Tright - Tr), 'all'));   % MD Right

% Consider the average MD for both the top and right boundaries
Zavg = mean([Zright Ztop]);

% Store the time required to solve
TimeEnd = toc(TimeStart);

end

% NESTED FUNCTION DEFINING MATRICES ---------------------------------------
function [A, B, C, K] = funSys(Ctx)

% RETRIEVE CONSTANTS ------------------------------------------------------
rhoCp    = Ctx.rhoCp;
```

```
k         = Ctx.k;
h         = Ctx.h ;
kg        = Ctx.kg;
Lx        = Ctx.Lx;
Ly        = Ctx.Ly;
R         = Ctx.R ;
Nx        = Ctx.Nx;
Ny        = Ctx.Ny;

Nx        = Ctx.Nx;
Ny        = Ctx.Ny;

% INITIALIZATIONS ------------------------------------------------------
dx  = Lx / ( Nx - 1 );    % differential - x
dy  = Ly / ( Ny - 1 );    % differential - y
x   = 0 : dx : Lx;        % discrete x axis, x(1) = 0, x(Nx) = Lx
y   = 0 : dy : Ly;        % discrete y axis, y(1) = 0, y(Ny) = Ly
N   = Nx * Ny;            % total number of unknowns (nodes)
vln = 1.0e8;              % very large number
A = sparse( 1, 1, 1, N, N, 5 * N ); % sparse linear system matrix
B = sparse( N, 2 );                 % input matrix

% C MATRIX -------------------------------------------------------------
C = sparse(2, N);        % output matrix
C(1, Ctx.node) =  1;     % Specify which node to consider

% GAIN MATRIX ----------------------------------------------------------
K = [ kg  0.0  ;         % gain matrix
    0.0 1.0 ];

% BOUNDARY CONDITIONS --------------------------------------------------

% For the left boundary
j = 1;
for i = 1 : Ny
    node  = Nx*(i-1)+j;
    nodeb = node + 1;
    A(node, node)   = vln*(-k/dx);
    A(node, nodeb)  = vln*(k/dx);
end

% For the right boundary
j = Nx;
for i = 1 : Ny
    node  = Nx*(i-1)+j;
    nodeb = node - 1;
    A(node, node)   = vln*(-k/dx-h);
    A(node, nodeb)  = vln*(k/dx);
    B(node, 2)      = vln*(h);
end

% For the bottom boundary
i = 1;
for j = 2 : Nx-1
    node  = Nx*(i-1)+j;
```

```
    nodeb = node+Nx;
    A( node,node)   = vln*(-k/dy);
    A( node,nodeb)  = vln*(k/dy);
end

% For the top boundary
i = Ny;
for j = 2 : Nx-1
    node  = Nx*(i-1)+j;
    nodeb = node - Nx;
    A( node, node)  = vln*(-k/dy-h);
    A( node, nodeb) = vln*(k/dy);
    B( node, 2)     = vln*(h);
end

% Considering internal nodes
Cleft   =  k / rhoCp / dx^2;
Cright  =  k / rhoCp / dx^2;
Cbottom =  k / rhoCp / dy^2;
Ctop    =  k / rhoCp / dy^2;
Ccenter = -k / rhoCp  * ( 2.0 / dx^2 + 2.0 / dy^2 );

for i = 2 : Ny-1
    for j = 2 : Nx-1

        % current node:
        node = Nx * ( i - 1 ) + j;

        % elements of B:
        if sqrt( x( j )^2 + y( i )^2 ) <= R
            B( node, 1 ) = 1.0 / rhoCp;
        end

        % elements of A:
        node_left   = node -  1; % ( i    , j - 1 )
        node_right  = node +  1; % ( i    , j + 1 )
        node_bottom = node - Nx; % ( i - 1, j     )
        node_top    = node + Nx; % ( i + 1, j     )
        node_center = node     ; % ( i    , j     )

        A( node, node_left   ) = Cleft  ;
        A( node, node_right  ) = Cright ;
        A( node, node_bottom ) = Cbottom;
        A( node, node_top    ) = Ctop   ;
        A( node, node_center ) = Ccenter;

    end
end
end
```

A.4 – Discrete Optimization for Mean deviation Calculator

```
% Discrete Optimization for Mean_Deviation.m
%
% MECH.5410 - Advanced Heat Transfer
% Project 3
% Nicholas McLaughlin
% 11/9/2020

% PURPOSE ----------------------------------------------------------------
% This code evaluates the function Mean_Deviation.m over a variety of
% thermocouple positions. The ambition is to determine which point will
% result in the greatest minimization of variability for the top and right
% boundaries.

% CODE DESCRIPTION -------------------------------------------------------
% This code makes use of a simple iterative optimization algorithm for
% discrete systems. This algorithm was created and designed by the author.
% Functionally, it considers the discretized 2D domain and divides it into
% 3x3 squares (i.e., a 27 x 9 matrix would have 9x3 squares). In the center
% of each square, the middle position is computed in Mean_Devaiation.m to
% get a reference as to the variance magnitude in that vicinity.
%
% After all the centers of each square has been evaluated, the code
% investigates the smallest variance. At this point, the entire 3x3 square
% is investigated. It is assumed that the minimum value is located within
% this square.
%
% Note, there are some portions of the domain that are not considered. For
% example, if the x-domain is discretized into 28 points, only the last 27
% will be considered. This is not a major concern because the purpose of
% this code is to generate a general idea of where the minimum variance is.
% The actual minimum can then be extrapolated and evaluated by hand.
%
% Following these calculations, the distribution of variance as a function
% of position is plotted. The variances for the top, right, and average are
% all plotted.
%
% This function also loops over a selection of k_g values. The data
% obtained is saved automatically such that it can be processed later.
% Note, this process takes quite a while. In its current configuration, it
% must compute 720 total calculations at ~5 seconds each. This ends up
% being a whole hour of computation time.

% MODEL DESCRIPTION------------------------------------------------------
% NOTE: This section is taken from Professor Trelles' code since it is
% nearly identical in functionality:
%
% Feedback control of a flat plate with a heating element.
%
% The temperature distribution T(x,y, t) through a rectangular domain of
% size Lx x Ly, with volumetric heat capacity rhoCp, thermal conductivity
% k, and volumetric heat generation g is described by:
%
%           rhoCo * dT/dt = k * d^2T/dx^2 + k * d^2T/dy^2 + g
%
% The term g( t ) is positive only within the conductor region - the
```

```
% control parameter, and zero elsewhere in the domain.
%
% The goal of the control approach is, given some (constant) reference
% temperature Tr and some varying Tinf(t), adjust the value of g such
% that the value of temperature at the point p (upper-right corner) Tp
% approaches Tr. The practical motivation of the problem is to maintain the
% top boundary (y = Ly) close to the reference temperature Tr.
%
% The initial condition for the problem is: ==============================
%
%             T = Tinitial, 0 < x < Lx, 0 < y < Ly, t = 0
%
% State-Space representation of the dynamic system:
%
%    dT/dt = A*T + B*u       ... (1)
%      y   = C*T             ... (2)
%      u   = K*(r - y)       ... (3)
%
% T: temperature of nodal values of temperature
% A: system matrix
% B: input matrix
% y: output vector,    y = [Tp; 0]
% C: output matrix,    C = [delta(ip, jp)], ip, jp: node number of p
% u: control vector,   u = [g; Tinf]
% K: gain matrix,      K = [kg 0; 0 1], (g = kg*(Tr - Tp))
% r: reference vector, r = [Tr; Tinf]
%
% Solution:
%
% T^(n+1) = (I - dt*(A - B*K*C))\(T^n + dt*B*K*r )
%
% Finite Difference Method (FDM) approximation:
% Physical domain:
%        y
%        ^
%        |
% (0, Ly)|               (top)           (Lx, Ly)
%        o---------------------------------o P <-- location of thermocouple
%        |                                 |
%        |                                 |
%        |                                 |
%        |                                 |
%        |                                 |
%        |                                 |
%  (left)|____                             |(right)
%        |    \__   (radius R)             |
%        |       \_                        |
%        |         \_                      |
%        | conductor \                     |
%        |             |                   |
%        o---------------------------------o ----> x
%  (0, 0)              (bottom)          (Lx, 0)
%
% Index domain:
%
%  (1, 1)            (bottom)         (1, Nx)
%        o---------------------------o -----> j (equivalent to x axis)
```

```
%         |                             |
%         |                             |
%         |                             |
%         |            i-1,j            |
%         |              |              |
%   (left)|    i,j-1 - i,j - i,j+1      |(right)
%         |              |              |
%         |            i+1,j            |
%         |                             |
%         |                             |
%         |                             |
%         o-----------------------------o
%   (Ny,1)|             (top)          (Ny, Nx)
%         |
%         i (equivalent to y axis)

% BOUNDARY CONDITIONS -----------------------------------------------------
% The boundary conditions are as follows for each edge:
%
% Bottom:       k * dT/dx = 0
% Left:        -k * dT/dx = 0
% Top:          k * dT/dy = h*(T -Tinf(t))
% Right:       -k * dT/dy = h*(T -Tinf(t))
%
% Essentially, this means there is no heat transfer at the bottom and left
% boundaries but there is convective heat transfer at the top and right
% boundaries. This is consistent with the requirements of the analysis.

% BEGINNING OF CODE -------------------------------------------------------
clear
close all

% Define k_g values to be considered throughout the loop
BIGG = [1.4e4 1.6e4 1.8e4 2.0e4 2.2e4];
KG = string(BIGG);

% BEGINNING OF LOOP -------------------------------------------------------
for w = 1:5
    kg = BIGG(w);

    Lx       = 0.020;     % [m]       length of domain along x
    Ly       = 0.010;     % [m]       length of domain along y
    tfinal   = 1e2;       % [s]       final time (1e2)
    Nx       =   51;      % number of nodes along x (101, 51)
    Ny       =   25;      % number of nodes along y (50, 25)
    x = linspace(0,Lx,Nx);
    y = linspace(0,Ly,Ny);

    % Specify Search Domain
    xd = [0.0 0.02];    % Make sure that x(2) = Lx in Mean_Deviation.m
    yd = [0.0 0.01];    % Make sure that y(2) = Ly in Mean_Deviation.m

% INITIAL CALCULATIONS ----------------------------------------------------
    % Determine the size of x and y
    [~,xn] = size(x);
```

```
    [~,yn] = size(y);

    % Calculate which matrix position corresponds with the cartesian
    % coordinate specified
    xld = round(Nx*(xd(1)/Lx));
    yld = round(Ny*(yd(1)/Ly));

    % Create preallocated matrices for faster computation
    T = zeros(Nx,Ny);
    zavg  = zeros(Nx,Ny);
    ztop = zeros(Nx,Ny);
    zright  = zeros(Nx,Ny);
    avgtop = zeros(Nx,Ny);
    avgright = zeros(Nx,Ny);
    sigtop = zeros(Nx,Ny);
    sigright = zeros(Nx,Ny);
    TimeEnd = zeros(Nx,Ny);

    % Calculate how many 3x3 squares are within the matrix
    xsquares = round((Nx-xld)/3);
    ysquares = round((Ny-yld)/3);

    % NOTIFICATION ------------------------------------------------------
    % Write to the screen the estimated calculations and time of completion
    Computations = xsquares*ysquares + 8;
    write = sprintf('Total Number of Computations: %.0f \n',Computations);
    fprintf(write);

    % This estimated time is based on the computer used to develop this code.
5
    % seconds was the average time of computation, if it varies on a
different
    % PC, please update the coefficient below accordingly
    EstTime = Computations*5;
    write = sprintf('Estimated time: %.0f [s]\n',EstTime);
    fprintf(write);
    EstTime = EstTime/60;
    write = sprintf('Estimated time: %.0f [min]\n',EstTime);
    fprintf(write);

% SCAN DOMAIN -------------------------------------------------------------
    % Preallocate matricies for storing the scanned data
    zIndex = zeros(xsquares,ysquares);
    zIndexTop = zIndex;
    zIndexRight = zIndex;

    % Loop to scan for the data within the domain
    for i = 1:xsquares
        for j = 1:ysquares
            % Start timer
            tic

            % Convert square indicies to the matrix discretization
            xi = Nx - (3*i - 2);
            yi = Ny - (3*j - 2);
```

```
            % Mean_Deviation solver
            [zavg(xi,yi),ztop(xi,yi),zright(xi,yi),avgtop(xi,yi),...
                avgright(xi,yi),sigtop(xi,yi),sigright(xi,yi),...
                TimeEnd(xi,yi)] = Mean_Deviation(xi,yi,Lx,Ly,Nx,Ny,kg);

            % Save data from the solution outside of the loop
            zIndex(i,j) = zavg(xi,yi);
            zIndexTop(i,j) = ztop(xi,yi);
            zIndexRight(i,j) = zright(xi,yi);

            % Stop timer & display time. This is primarily used to ensure the
            % code is operating
            toc
        end
    end

    % Find where the lowest mean deviation is in the resulting matrix
    [M,I] = min(zIndex(:));
    [I_row, I_col] = ind2sub(size(zIndex),I);

    % Specify the what points in the zavg matrix will have the lowest mean
    % deviation
    Px = Nx - (3*I_row - 2);
    Py = Ny - (3*I_col - 2);

% FURTHER INVESTIGATION ---------------------------------------------------
    % Notify user the initial scan is complete
    disp('Located area to investigate further')
    load gong
    sound(y,Fs)

    % Second loop to calculate the 9 points around (and including) the
    % smallest value found from the scan
    for i = (Px-1):(Px+1)
        for j = (Py-1):(Py+1)
            % Start timer
            tic
            % Update the exisiting matrices with the solutions for the
specific
            % points
            [zavg(i,j),ztop(i,j),zright(i,j),avgtop(i,j),...
                avgright(i,j),sigtop(i,j),sigright(i,j),...
                TimeEnd(i,j)] = Mean_Deviation(i,j,Lx,Ly,Nx,Ny,kg);
            % Stop timer & display time. This is primarily used to ensure the
            % code is operating
            toc
        end
    end
    filename = strcat('KG',KG(w),'.mat');
    save(filename);
end
```

A.5 – Data Analysis

```
% Analysis of Data from Iterative_Method.m
%
% MECH.5410 - Advanced Heat Transfer
% Project 3
% Nicholas McLaughlin
% 11/9/2020

% PURPOSE ----------------------------------------------------------------
% This code processes the immense amount of data from Iterative_Method.m.
% The reason these commands are not integrated into the code initially is
% due to the changing asthetic and numerical requirements for considering
% the data. Instead of waiting for ~80 minutes of code processing, this
% code can analyze all the data generated in under 5 seconds.

% CODE DESCRIPTION -------------------------------------------------------
% This series of code loads the workspaces saved after each run of
% Iterative_Method.m for different values of k_g. There are several results
% desired from this processing:
%   1. Find the mean deviation at 6 points
%   2. Find where the minimum mean deviation is in the matrix z_avg
%   3. Plot the mean deviation surface as a function of thermocouple
%      position
%       a.) For the average mean deviation
%       b.) For the top boundary mean deviation
%       c.) For the right boundary mean deviation
%   4. Plot the mean deviation of the 6 points as a function of k_g
%
% Necessarily, this code is repetitive. Since there will be 5 sections of
% code that are nearly identical, only the first section will be annotated
% with detail. The same annotations apply equally well to the subsequent 4
% sections.
%
% The 6 points being considered are drawn below :
%
% ____________________________________________________________________
% |(2)                                                          (3)|
% |                                                                |
% |                                                                |
% |                                                                |
% |                                                                |
% |                                                                |
% |                                        (4)                     |
% |                                                                |
% |                                                                |
% |                                                                |
% |                                                                |
% |                                                                |
% |(1)_____________________________________________(5)_____________(6)|


% BEGIN CODE -------------------------------------------------------------
clear
close all
```

```
% Preallocate the matrix z to store the MD values for the 6 points
MD = zeros(6,5);

% Create the k_g vector used to order each data set
g = [1.4e4 1.6e4 1.8e4 2.0e4 2.2e4];

% LOAD FIRST DATA SET ----------------------------------------------------
load('KG14000.mat')

% Measure the size of the matrix zIndex
[m,n] = size(zIndex);

% Extract the mean deviation for each of the 6 points
MD(1,1) = zIndex(end,end);                      % Lower left corner
MD(2,1) = zIndex(end,1);                        % Upper left corner
MD(3,1) = zIndex(1,1);                          % Upper right corner
MD(4,1) = zIndex(round(0.25*m),round(0.5*n));   % Middle-right
MD(5,1) = zIndex(end,round(0.5*n));             % Middle right edge
MD(6,1) = zIndex(1,end);                        % Lower right corner

% Search through the zavg matrix and change all the zero elements to 10
% This is necessary because we want to know what the minimum value is of
% the data collected - and we purposefully did not fill the zeros matrix
% initially
[m,n] = size(zavg);
for i = 1:m
    for j = 1:n
        if zavg(i,j) == 0
            zavg(i,j) = 10;
        end
    end
end

% Find where the minimum mean deviation value is
[~,I] = min(zavg(:));
[I_row, I_col] = ind2sub(size(zavg),I);

% Conver the columns and rows into the physical distances
xdist = (I_row/Nx)*Lx;
ydist = (I_col/Ny)*Ly;

% Display the results to screen
write = 'Minimum for k_g = 1.4e4 \n x = %.6f [m] y = %.6f [m] \n \n';
fprintf(write,xdist,ydist)

% Create indicies for plotting
i = 1:xsquares;
j = 1:ysquares;
xi = (Nx - (3*i - 2))*(Lx/Nx);
yi = (Ny - (3*j - 2))*(Ly/Ny);

% Plot the mean mean deviation as a function of thermocouple position
figure
surf(yi,xi,zIndex);
xlabel('y-Dimension [m]')
```

```
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Deviation vs Thermocouple Position',...
    ' (k_g = 1.40 x 10^4) '})
view(138,-27)
daspect([1 1 200])

% Plot the top mean deviation as a function of thermocouple position
figure
surf(yi,xi,zIndexTop)
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Deviation vs Thermocouple Position (Top Edge)',...
    ' (k_g = 1.40 x 10^4) '})
view(138,-27)
daspect([1 1 200])

% Plot the right mean deviation as a function of thermocouple position
figure
surf(yi,xi,zIndexRight)
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Deviation vs Thermocouple Position (Right Edge)',...
    ' (k_g = 1.40 x 10^4) '})
view(138,-27)
daspect([1 1 200])

% LOAD SECOND DATA SET ------------------------------------------------------
% See data set 1 for detailed explanation of code
load('KG16000.mat')
[m,n] = size(zIndex);

MD(1,2) = zIndex(end,end);
MD(2,2) = zIndex(end,1);
MD(3,2) = zIndex(1,1);
MD(4,2) = zIndex(round(0.25*m),round(0.5*n));
MD(5,2) = zIndex(end,round(0.5*n));
MD(6,2) = zIndex(1,end);

[m,n] = size(zavg);
for i = 1:m
    for j = 1:n
        if zavg(i,j) == 0
            zavg(i,j) = 10;
        end
    end
end

% Create indicies for plotting
i = 1:xsquares;
j = 1:ysquares;
xi = (Nx - (3*i - 2))*(Lx/Nx);
yi = (Ny - (3*j - 2))*(Ly/Ny);
```

```
[~,I] = min(zavg(:));
[I_row, I_col] = ind2sub(size(zavg),I);
write = 'Minimum for k_g = 1.6e4 \n x = %.6f [m] y = %.6f [m] \n \n';
xdist = (I_row/Nx)*Lx;
ydist = (I_col/Ny)*Ly;
fprintf(write,xdist,ydist)

figure
surf(yi,xi,zIndex);
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({'Mean Deviation vs Thermocouple Position',...
    ' (k_g = 1.60 x 10^4) '})
view(138,-27)
daspect([1 1 200])

figure
surf(yi,xi,zIndexTop)
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({'Mean Deviation vs Thermocouple Position (Top Edge)',...
    ' (k_g = 1.60 x 10^4) '})
view(138,-27)
daspect([1 1 200])

figure
surf(yi,xi,zIndexRight)
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Deviation vs Thermocouple Position (Right Edge)',...
    ' (k_g = 1.60 x 10^4) '})
view(138,-27)
daspect([1 1 200])

% LOAD THIRD DATA SET ------------------------------------------------------
% See data set 1 for detailed explanation of code
load('KG18000.mat')
[m,n] = size(zIndex);

MD(1,3) = zIndex(end,end);
MD(2,3) = zIndex(end,1);
MD(3,3) = zIndex(1,1);
MD(4,3) = zIndex(round(0.25*m),round(0.5*n));
MD(5,3) = zIndex(end,round(0.5*n));
MD(6,3) = zIndex(1,end);

[m,n] = size(zavg);
for i = 1:m
    for j = 1:n
        if zavg(i,j) == 0
            zavg(i,j) = 10;
```

```
        end
    end
end

i = 1:xsquares;
j = 1:ysquares;
xi = (Nx - (3*i - 2))*(Lx/Nx);
yi = (Ny - (3*j - 2))*(Ly/Ny);

[~,I] = min(zavg(:));
[I_row, I_col] = ind2sub(size(zavg),I);
write = 'Minimum for k_g = 1.8e4 \n x = %.6f [m] y = %.6f [m] \n \n';
xdist = (I_row/Nx)*Lx;
ydist = (I_col/Ny)*Ly;
fprintf(write,xdist,ydist)

figure
surf(yi,xi,zIndex);
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({'Mean Deviation vs Thermocouple Position',...
    ' (k_g = 1.80 x 10^4) '})
view(138,-27)
daspect([1 1 200])

figure
surf(yi,xi,zIndexTop)
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Deviation vs Thermocouple Position (Top Edge)',...
    ' (k_g = 1.80 x 10^4) '})
view(138,-27)
daspect([1 1 200])

figure
surf(yi,xi,zIndexRight)
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Deviation vs Thermocouple Position (Right Edge)',...
    ' (k_g = 1.80 x 10^4) '})
view(138,-27)
daspect([1 1 200])

% LOAD FOURTH DATA SET ------------------------------------------------------
% See data set 1 for detailed explanation of code
load('KG20000.mat')
[m,n] = size(zIndex);

MD(1,4) = zIndex(end,end);
MD(2,4) = zIndex(end,1);
MD(3,4) = zIndex(1,1);
MD(4,4) = zIndex(round(0.25*m),round(0.5*n));
```

```
MD(5,4) = zIndex(end,round(0.5*n));
MD(6,4) = zIndex(1,end);

i = 1:xsquares;
j = 1:ysquares;
xi = (Nx - (3*i - 2))*(Lx/Nx);
yi = (Ny - (3*j - 2))*(Ly/Ny);

[m,n] = size(zavg);
for i = 1:m
    for j = 1:n
        if zavg(i,j) == 0
            zavg(i,j) = 10;
        end
    end
end

[~,I] = min(zavg(:));
[I_row, I_col] = ind2sub(size(zavg),I);
write = 'Minimum for k_g = 2.0e4 \n x = %.6f [m] y = %.6f [m] \n \n';
xdist = (I_row/Nx)*Lx;
ydist = (I_col/Ny)*Ly;
fprintf(write,xdist,ydist)

figure
surf(yi,xi,zIndex);
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Deviation vs Thermocouple Position',...
    ' (k_g = 2.00 x 10^4) '})
view(138,-27)
daspect([1 1 200])

figure
surf(yi,xi,zIndexTop)
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Deviation vs Thermocouple Position (Top Edge)',...
    ' (k_g = 2.00 x 10^4) '})
view(138,-27)
daspect([1 1 200])

figure
surf(yi,xi,zIndexRight)
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Deviation vs Thermocouple Position (Right Edge)',...
    ' (k_g = 2.00 x 10^4) '})
view(138,-27)
daspect([1 1 200])

% LOAD FIFTH DATA SET ----------------------------------------------------
```

```
% See data set 1 for detailed explanation of code
load('KG22000.mat')
[m,n] = size(zIndex);

MD(1,5) = zIndex(end,end);
MD(2,5) = zIndex(end,1);
MD(3,5) = zIndex(1,1);
MD(4,5) = zIndex(round(0.25*m),round(0.5*n));
MD(5,5) = zIndex(end,round(0.5*n));
MD(6,5) = zIndex(1,end);

[m,n] = size(zavg);
for i = 1:m
    for j = 1:n
        if zavg(i,j) == 0
            zavg(i,j) = 10e10;
        end
    end
end

i = 1:xsquares;
j = 1:ysquares;
xi = (Nx - (3*i - 2))*(Lx/Nx);
yi = (Ny - (3*j - 2))*(Ly/Ny);

[~,I] = min(zavg(:));
[I_row, I_col] = ind2sub(size(zavg),I);
write = 'Minimum for k_g = 2.2e4 \n x = %.6f [m] y = %.6f [m] \n \n';
xdist = (I_row/Nx)*Lx;
ydist = (I_col/Ny)*Ly;
fprintf(write,xdist,ydist)

figure
surf(yi,xi,zIndex);
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Deviation vs Thermocouple Position',...
    ' (k_g = 2.20 x 10^4) '})
view(138,-27)
daspect([1 1 200])

figure
surf(yi,xi,zIndexTop)
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Deviation vs Thermocouple Position (Top Edge)',...
    ' (k_g = 2.20 x 10^4) '})
view(138,-27)
daspect([1 1 200])

figure
surf(yi,xi,zIndexRight)
xlabel('y-Dimension [m]')
```

```
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Deviation vs Thermocouple Position (Right Edge)',...
    ' (k_g = 2.20 x 10^4) '})
daspect([1 1 200])
view(138,-27)

xsquares = round((Nx-xld)/3);
ysquares = round((Ny-yld)/3);

for i = 1:xsquares
    for j = 1:ysquares
        xii = Nx - (3*i - 2);
        yii = Ny - (3*j - 2);
        AvgRight(i,j) = avgright(xii,yii);
        AvgTop(i,j) = avgtop(xii,yii);
        SigRight(i,j) = sigright(xii,yii);
        SigTop(i,j) = sigtop(xii,yii);

    end
end

figure
surf(yi,xi,AvgRight)
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Temperature vs Thermocouple Position (Right Edge)',...
    ' (k_g = 2.20 x 10^4) '})
daspect([1 1 200])
view(138,-27)

figure
surf(yi,xi,AvgTop)
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Mean Temperature vs Thermocouple Position (Top Edge)',...
    ' (k_g = 2.20 x 10^4) '})
daspect([1 1 200])
view(138,-27)

figure
surf(yi,xi,SigRight)
xlabel('y-Dimension [m]')
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Standard Deviation vs Thermocouple Position (Right Edge)',...
    ' (k_g = 2.20 x 10^4) '})
daspect([1 1 200])
view(138,-27)

figure
surf(yi,xi,SigTop)
xlabel('y-Dimension [m]')
```

```
ylabel('x-Dimension [m]')
zlabel('Mean Deviation [°C]')
title({' Standard Devation vs Thermocouple Position (Top Edge)',...
    ' (k_g = 2.20 x 10^4) '})
daspect([1 1 200])
view(138,-27)

% FINAL PLOTTING ------------------------------------------------------------
% Consider each of the 6 points as a function of k_g
figure
hold on
plot(g,MD(1,:))
plot(g,MD(2,:))
plot(g,MD(3,:))
plot(g,MD(4,:))
plot(g,MD(5,:))
plot(g,MD(6,:))

legend('Point 1','Point 2','Point 3','Point 4','Point 5','Point 6')
xlabel('k_g')
ylabel('Mean Deviation')
title('Mean Deviation for Various Points as a Function of k_g')
```