Mason Rodriguez Rand and Chris Simotas 9/25/2023

ME 249 Project 1 Report

1. Introduction

Boiling heat transfer, a vital process in thermal management systems, presents unique challenges in varying gravitational environments. On Earth, gravity aids in moving vapor away from heated surfaces, sustaining the boiling process. However, in microgravity or zero gravity conditions, this mechanism diminishes. Vapor accumulates near the heated surface, rather than moving away, compromising the heat transfer efficiency to the remaining volume of fluid. This puts system performance and equipment integrity at risk.

One intriguing finding, however, is that water/2-propanol mixtures exhibit enhanced nucleate boiling heat transfer under reduced gravity conditions. This enhancement is attributed to Marangoni forces, forces driven by surface tension gradients from fluid concentration differences, which induce liquid motion towards the heated surface.

In this project report, we explore the interplay of gravity and Marangoni effects in boiling water/2-propanol mixtures by leveraging machine-learning tools to unveil the intricate dependencies across several variables. Specifically, we examine the variations in heat flux (q'') under across a spectrum of gravitational accelerations (g), surface conditions (γ), wall superheats (T_w-T_{sat}), and pressures (P). By creating a model, we hope to reach a better understanding of boiling heat transfer for water/2-propanol mixtures in various gravitational environments.

Additionally, to continue our investigations into heat transfer, we explored the various parameters that affect the heat transfer performance of heat pipes. Heat pipes, with their complex evaporator fin design, are critical tools used in computer electronics to regulate internal temperatures. By using machine learning tools to evaluate their heat flux (q'') performance across variables such as internal air temperature ($T_{a,in}$) and external air temperature ($T_{a,out}$), we hope to further discover the impact of heat pipe's unique design.

2. Separation of Tasks

Here is a list of the separation of tasks for Project 1 (C = Chris, M = Mason):

- Project Implementation
  - Read and discuss project description (C, M)
  - Create strategy and timeline for project completion and separation of tasks (C, M)
- Task 1
  - Alternating typist and speaker/alternate relevant document referencer roles (C, M)
  - Task 1.2 & 1.3 follow-up and refinement of additional code drafts
    - Retesting n1, n2, n3 values to find additional patterns (C)
    - Surface plots + follow-up modifications in task 1.3 (M)

- Task 2
  - Alternating typist and speaker/alternate relevant document referencer roles (C, M)
  - Log-log and non-log-log plots heat transfer rate predicted vs data with circular and square data points (C)
  - Surface plots + follow-up modifications in task 2.2 (M)
- Project Report
  - Outline (C)
  - Introduction (C)
  - Separation of Tasks (M)
  - Task 1 Code Modifications (M)
  - Task 1.1 and 1.2 Analysis and Discussion (C)
  - Task 1.3 Analysis and Discussion (M)
  - Task 2 Code Modifications (C)
  - Task 2 Analysis and Discussion (C, M)
  - Discussion of Raw vs. Dimensionless Data Analysis (C, M)

3. Task 1

3.1. Task 1.1 & 1.2 Code Modifications Summary

1. First we Copy+Pasted **CodeP1.1F23** and ran it. We confirmed the output is the same as the given data.
2. Next, we inserted code to construct a log-log plot of heat flux vs wall superheat:

```
# PLOTTING LOG OF HEAT FLUX AND WALL TEMP

x = [item[1] for item in lydata[: 29]]
y = [item[0] for item in lydata[: 29]]

# defining regression line constants
m, b = numpy.polyfit(x, y, 1)

#plotting log(data) and regression line
plt.scatter(x, y)
plt.plot(x, m*(numpy.array(x)) + b)

plt.title("Log-Log Plot of Heat Flux vs Wall Superheat")
plt.xlabel("Wall Superheat log(T$_W$-T$_{sat}$ (˚C))", labelpad = 10)
plt.ylabel("Heat Flux log(W/cm$^2$)", rotation = 90, labelpad = 10)
plt.legend(["Regression Line: y = {:.2f}x + {:.2f}" .format(m, b)])

plt.xlim(xmax = 5.5, xmin = 3.25)
plt.ylim(ymax = 5.5, ymin = 3.25)
plt.show()
```

3. We next added **CodeP1.2 F23**, varied the constants n1, n2, n3, modified our number of generations or the “NGEN” variable, and also changed the perturbation rate multiple times. Exact modification values are listed in **Table 1** below. In code, these looked like the following, with different values assigned to the given variables in each case. The red highlighted “.09” values indicate where the perturbation rate was modified:

```
#set program parameters
NGEN = 12000 #number of generations (steps)

'''guesses for initial solution population'''
n0i = -1.0
n1i = .0014
n2i = 3
n3i = .04
n4i = 1.4
n5i = .100

if (numpy.random.rand() < 0.5):
ntemp[nkeep+j+1][1] = n[nmate1][1]*(1.+0.09*2.*(0.5-numpy.random.rand()))
# property 1, mutation added
else:
ntemp[nkeep+j+1][1] = n[nmate2][1]*(1.+0.09*2.*(0.5-numpy.random.rand()))

if (numpy.random.rand() < 0.5):
ntemp[nkeep+j+1][2] = n[nmate1][2]*(1.+0.09*2.*(0.5-numpy.random.rand()))
# property 2, mutation added
else:
ntemp[nkeep+j+1][2] = n[nmate2][2]*(1.+0.09*2.*(0.5-numpy.random.rand()))

if (numpy.random.rand() < 0.5):
ntemp[nkeep+j+1][3] = n[nmate1][3]*(1.+0.09*2.*(0.5-numpy.random.rand()))
# property 3, mutation added
else:
ntemp[nkeep+j+1][3] = n[nmate2][3]*(1.+0.09*2.*(0.5-numpy.random.rand()))
```

4. Finally, we added code to construct a log-log plot of the measured vs predicted heat flux values as well as assigned a linear regression line to their values and determine the rms deviation of the data from the predictions. The code for which is as follows:

```
### PLOTTING BEST N1, N2, N3 CONSTANTS ###

# Best Constants
n1_best = 0.00164596567414208
n2_best = 2.985685746771903
n3_best = 0.044561714504267345

# Initialize Arrays
qpppred_best = [[0.0]]
qppdata_best = [[0.0]]
for i in range(ND-1):
qpppred_best.append([0.0])
qppdata_best.append([0.0])

# Calculate predicated and data values both normal and LOG
for i in range(ND):
qpppred_best[i] = n1_best*(ydata[i][1]**n2_best) *
((ydata[i][2])**n3_best)
qppdata_best[i] = ydata[i][0]

lqpppred_best = [math.log(d) for d in qpppred_best]
lqppdata_best = [math.log(d) for d in qppdata_best]

# Create Regression Line
m, b = numpy.polyfit(lqpppred_best, lqppdata_best, 1)

# Plot Log-Log Plot
plt.scatter(lqpppred_best, lqppdata_best, marker = 'o', linestyle = '-')
plt.plot(lqpppred_best, m*(numpy.array(lqpppred_best)) + b)
#plt.loglog()

plt.title('Log-Log Plot of Heat Flux Predicted Vs. Measured (Best Fit)')
plt.xlabel('Predicted Heat Flux log(W/cm$^2$)\n n1 = 0.00165 n2 = 2.99 n3
= 0.0446', labelpad = 10)
plt.ylabel('Measured Heat Flux log(W/cm$^2$)', labelpad = 10)
plt.legend(["Regression Line: y = {:.2f}x + {:.2f}" .format(m, b)])
plt.xlim(xmax = 6.1, xmin = 3)
plt.ylim(ymax = 6.1, ymin = 3)
```

```
### Output RMSD Value ###
diff = (numpy.array(qppdata_best) -
numpy.array(qpppred_best))/numpy.array(qppdata_best)
rms_dev = numpy.sqrt(numpy.mean(diff**2))
print("RMS Relative Deviation between qpppred and qppdata: " +
str(rms_dev) + "\n")
```

3.2. Task 1.1 & 1.2 Analysis and Discussion

During our initial assessment of the dataset for Task 1, we created a log-log plot of heat flux (W/cm^2) versus wall superheat (°C) for when gravity equaled 0.098 and 9.8 m/s^2 as shown in **Figure 1**. The plotted data, with the help of a regression line, shows that there is a slope greater than 1 between the wall superheat and heat flux. This suggests that the logarithm of the dependent variable, heat flux, increases at a faster rate than the logarithm of the independent variable, wall superheat.

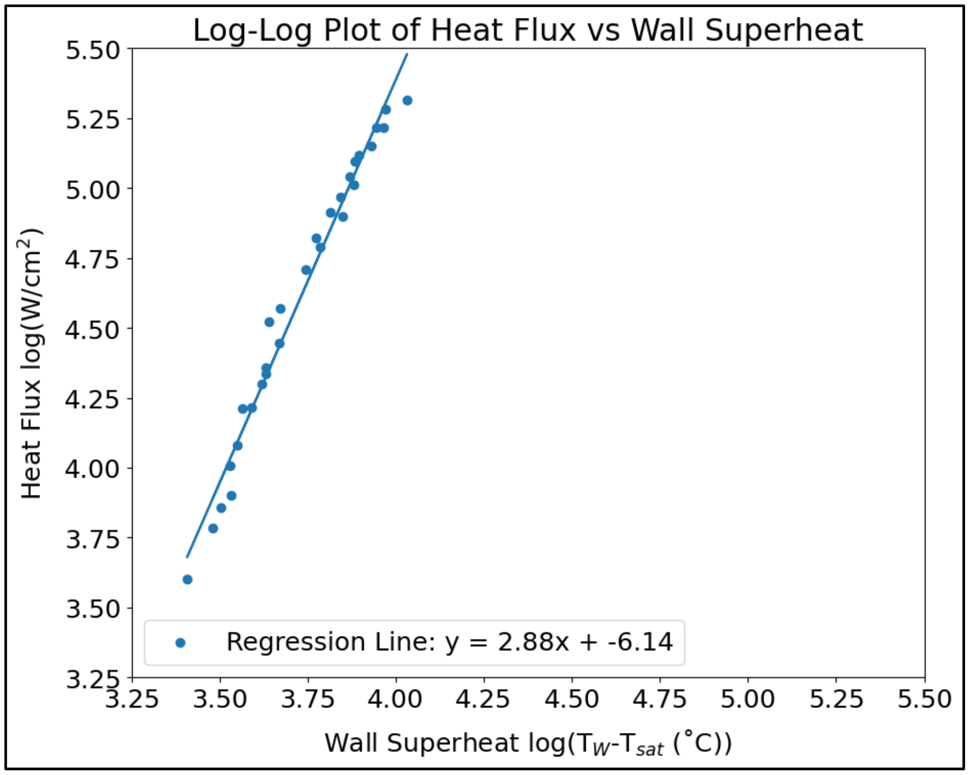


Figure 1. Log-Log Plot of Heat Flux vs. Wall Superheat when g = 9.8 & 0.098

With our first assessment complete, we next ran a genetic algorithm to optimize our hypothesized relation for the dependence of heat flux on wall superheat and gravity:

$$q'' = n_1 \, (T_w - T_{sat})^{n2} \, g^{n3}$$

Specifically, we aimed for the genetic algorithm to optimize the constants n_1, n_2, and n_3 that would best suit the relationship between the independent and dependent variables. Our fitness measurement for each generation equated to the mean of our total error function $(F_{err})_{mean}$, which calculates the absolute fractional error of each data point. We were attempting to minimize $(F_{err})_{mean}$ below 0.03. While running the algorithm, we explored five different sets of initial guesses as shown in **Table 1**.

Table 1:

Guess Set	**Initial Guess** n_1 n_2 n_3	**Best Fit of Round** n_1 n_2 n_3	**# of Gen**	**Perturbation Rate**	**Time Avg** $(F_{err})_{mean}$	**Min** $(F_{err})_{mean}$
1	3.00 10.0 5.00	0.0774 2.37 0.0111	6000	50%	0.414	0.0340
2	0.00200 3.00 0.100	0.000312 3.91 0.800	6000	9%	0.572	0.503
3	0.00200 3.00 0.100	0.000941 3.14 0.059	6000	30%	0.135	0.0294
4	0.001 2.00 1.00	0.00105 2.97 0.332	6000	9%	0.205	0.139
5	0.001 2.00 1.00	0.00165 2.99 0.0446	6000	30%	0.0613	0.0286

Summary of Initial Guesses and Model Results for Three Variable Model

By manipulating the *n* constants and perturbation rate for mutation, we gained better knowledge about several local min solutions and their associated constants. Specifically, we found that a higher perturbation mutation rate led to better success in finding viable solutions (generations with fractional errors lower than 0.03). We were able to rule out any large constant values (greater than 3) with Guess Set 1. Using a high perturbation rate allowed the algorithm to test over a larger range of solutions and ultimately led to better fitness scores. After optimizing, it became apparent that the constant values were much smaller than we had initially guessed, especially for n_1 and n_3.

Further refinement of the perturbation rate helped us rule out other constant values which experienced lower fitness. For example, we found that initial guesses greater than 0.1 for n_3 did not lead to solutions with strong fitness unless there was a large perturbation rate (See Sets 2 and 4 in **Table 1**). As a result, we concluded that n_3 should be relatively close to 0 for more optimal solutions. The same applied for n_1 as our best solutions all experience n_1 less than 0.002.
The most surprising detail of our experimentation was the consistency around the values for n_2 during optimization. **Figure 2** shows the variation in values for each of the three *n* constants when the initial guesses were placed near our best fit solution. The n_2 constant remained extremely consistent compared to n_1 and n_3. From this we concluded that there are likely several other solutions for the n constants near our best fit solution which all experience relatively similar n_2 values.

It is also important to note that unless our initial guesses were close to the best fit solution or there was a large perturbation rate, the algorithm had difficulty quickly converging on a solution as it can be seen in **Table 2**. Most of the time, the algorithm did not converge on a solution in our acceptable range. However, if the guesses were close enough, we saw the algorithm converge pretty quickly as it can be seen by aFerrmeanavgn in **Figure 3**.

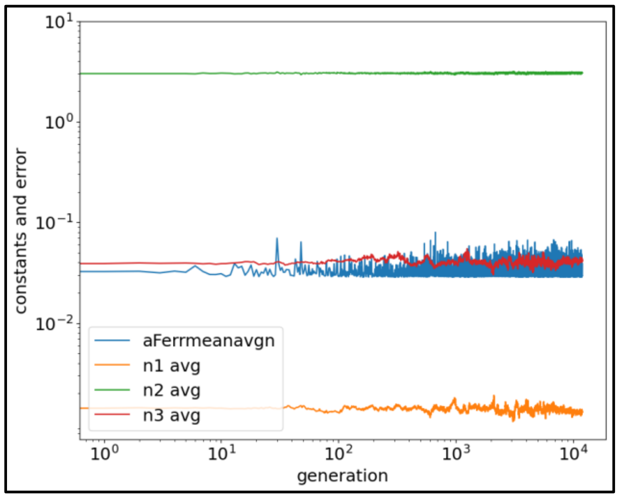


Figure 2. Variation in Constant Values during Best Fit Model Optimization for 3 Constant Model

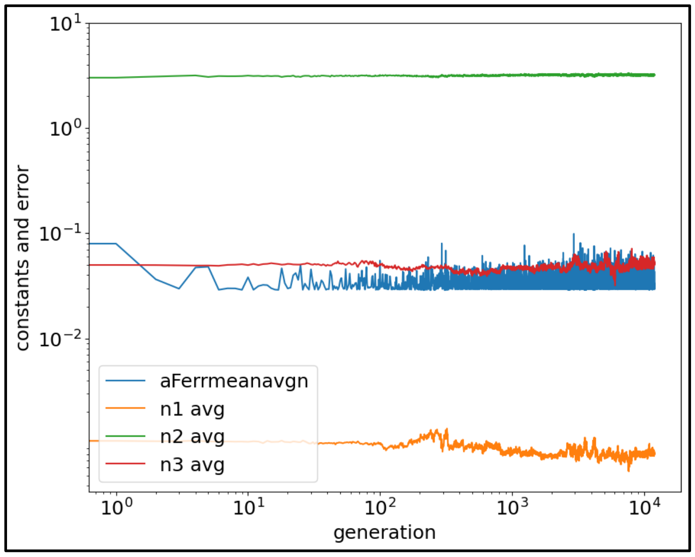


Figure 3. Variation in Constant Values When Initial Guesses are Near Final Constants

After several rounds of experimenting, we converged on our best fit solution shown in the equation below:

$$q'' = 0.00165(T_w - T_{sat})^{2.99}\, g^{0.0466}$$

It is clear from this model that heat flux is highly dependent on wall superheat given the converged value of 2.99 for n_2. This was anticipated as we expect the heat going into a system to affect the amount of heat leaving it. Conversely, gravity has a fairly minimal, though non-zero effect on heat flux. Based on this, it seems to indicate that the Marangoni forces present in water/2-propanol mixtures create enough fluid motion to promote heat transfer to counteract the negative effects of gravity.

Overall, our model performed decently well when comparing our predicted heat flux values to the measured values as shown in **Figure 4**. Specifically, the slope of the graph's regression line reveals that the predicted values are slightly lower than their measured value counterparts. To dive a bit deeper, we calculated the root mean square deviation (RMSD) between the measured and predicted heat flux and compared it to the uncertainty of the measured heat flux in **Table 2**.

Our model's RMSD of 17% is larger than the measured heat flux's uncertainty of 10%, signifying that the average deviation in the model's predictions is larger than the range of expected variations in the measured data. Thus, the model's predictions exhibit a level of error that exceeds what can be attributed to uncertainty in the data. Considering this, it will be difficult to make accurate statements of predicted trends from this fitted model since the model is inducing its own variability. We may have experienced a lower RMSD if we instead trained the model over more data, or continued to reach better *n* constant values through a similar guessing process as exhibited in **Table 1**. On the other hand, there may also be other independent variables or parameters not in our model that need to be included when calculating q".

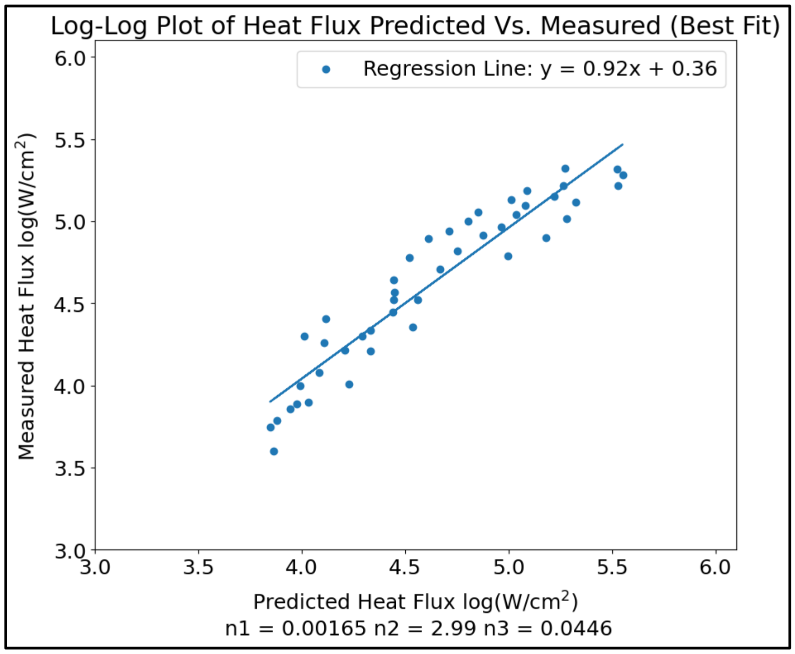


Figure 4. Log-Log Plot of Heat Flux Predicted vs Measured with Best Fit

Table 2. Calculated Error of Best Fit Model (3 Variable)

	Min(F_{err})$_{mean}$	Measured Heat Flux Uncertainty	Root Mean Square Deviation Between Measured and Predicted Heat Flux
Best Fit Model (3 Variable)	0.0286	10%	17.1%

3.3. Task 1.3 Code Modifications

1. First we Copy+Pasted our combined CodeP1.1F23 and CodeP1.2F23.
2. Next, we added more data to the model by uncommenting the following two code blocks to append to ydata:

```
#'''
ydata.append([42.4, 29.7, 19.6, 1.79, 5.5])
... (more data here, removed for brevity)
ydata.append([205.0, 47.9, 19.6, 1.79, 5.5])
#'''
#'''
ydata.append([77.0, 41.5, 9.8, 0.00, 7.0])
... (more data here, removed for brevity)
ydata.append([207.5, 50.9, 0.098, 1.71, 5.5])
#'''
```

3. ND and NS, the number of data vectors in ydata (rows), and the number of DNA strands were changed from 45 to 77 to reflect the additional added data as follows (red highlighted “77s” are the modifications:

```
#Parameters for Evolution Loop
#set data parameters
ND = 77 #number of data vectors in array
DI = 5 #number of data items in vector
NS = 77 #total number of DNA strands
```

4. The following sections were modified from the original code to reflect the appropriate error and heat flux quantities for the five constant model (modifications bolded and highlighted):

```
'''CALCULATING ERROR (FITNESS)'''
for i in range(ND):
Ferr[i] = n[i][0]*lydata[i][0] + math.log(n[i][1]) + n[i][2]*lydata[i][1]
Ferr[i] = Ferr[i] + n[i][3]*math.log( ydata[i][2] )

Ferr[i] = Ferr[i] +
n[i][3]*math.log(ydata[i][2]+n[i][4]*9.8*ydata[i][3])+n[i][5]*lydata[i][4]
```

This reflects the change of the error equation in the three vs five constant models, or from **equation 1.3** to **equation 1.7** in the project document.

```
''' CALCULATING MEAN ERROR FOR POPULATION'''
for i in range(ND):
Ferravgn[i] = -1.*lydata[i][0] + math.log(n1avg[k]) +
n2avg[k]*lydata[i][1]
Ferravgn[i] = Ferravgn[i] + n3avg[k]*math.log( ydata[i][2])

Ferravgn[i] = Ferravgn[i] +
n3avg[k]*math.log(ydata[i][2]+n4avg[k]*9.8*ydata[i][3])+n5avg[k]*lydata[i]
[4]
```

This reflects the change of the *average* error equation in the three vs five constant models, or from **equation 1.4** in the project document to **equation 1.8**.

```
for i in range(ND):
qpppred[i] = n1min*(ydata[i][1]**n2min) * ((ydata[i][2])**n3min)

qpppred[i] = n1min*(ydata[i][1]**n2min) *
((ydata[i][2]+n4min*9.8*ydata[i][3])**n3min)*ydata[i][4]**n5min
```

Finally, this modification reflects the change of the predicted q" value in the three vs five constant models. Namely changing from **equation 1.1** to **equation 1.5** in the project document.

5. The following lines were uncommented to include n_4 and n_5 in mating. Without these lines, no mutations would be generated from generation to generation for n_4 and n_5 which would likely slow progress to a solution:

```
if (numpy.random.rand() < 0.5):
ntemp[nkeep+j+1][4] = n[nmate1][4]*(1.+0.09*2.*(0.5-numpy.random.rand()))
# property 4, mutation added
else:
ntemp[nkeep+j+1][4] = n[nmate2][4]*(1.+0.09*2.*(0.5-numpy.random.rand()))

if (numpy.random.rand() < 0.5):
ntemp[nkeep+j+1][5] = n[nmate1][5]*(1.+0.09*2.*(0.5-numpy.random.rand()))
# property 5, mutation added
else:
ntemp[nkeep+j+1][5] = n[nmate2][5]*(1.+0.09*2.*(0.5-numpy.random.rand()))
```

6. The print functions were modified to include the last and minimum n_4 and n_5 values as well as their time averages as shown here:

```
print('ENDING: pop. avg n1-n5,aFerrmean:', n1avg[k], n2avg[k], n3avg[k],
n4avg[k], n5avg[5], aFerrmeanavgn[k])
print('MINUMUM: avg n1-n5,aFerrmeanMin:', n1min, n2min, n3min, n4min,
n5min, aFerrmeanavgnMin)
print('TIME AVG: avg n1-n5,aFerrmean:', n1ta, n2ta, n3ta, n4ta, n5ta,
aFerrta)
```

7. The legend of the plot was modified to include the n_4 and n_5 values by adding the lines:

```
plt.plot(gen, n4avg)
plt.plot(gen, n5avg)
plt.legend(['aFerrmeanavgn', 'n1 avg', 'n2 avg', 'n3 avg', 'n4 avg', 'n5
avg'], loc='upper right')
```

8. The initial guesses for n_1-n_5 were changed to 0.000476, 3.028, 0.2249, 1.054, 0.217 respectively.

3.4. Task 1.3 Analysis and Discussion

Before, our model of heat flux was defined by **equation 1.1** as follows: $\mathbf{q'' = n_1(T_w - T_{sat})^{n2}\, g^{n3}}$. In tasks 1.1 and 1.2, we sought to find the best constants n_1, n_2, and n_3 to define a model that relates heat flux to superheat and gravitational acceleration. Now, we take into account effects from the pressure of the surrounding environment and the surface tension parameter of the fluid. We define the model as $\mathbf{q'' = n_1(T_w - T_{sat})^{n2}\, (g + n_4 g_{en}\gamma)^{n3}\, P^{n5}}$ and seek to find the best n_4 and n_5 in addition to the n_1, n_2, and n_3 constants to minimize our error function. The error function is also now defined instead to include the resulting error contributions from n_4 and n_5 as in **equation 1.7** in the project document. We set a goal as outlined in the project to reach **Min** $(\boldsymbol{F_{err}})_{\boldsymbol{mean}}$ less than 0.03. Our four guesses at n values are displayed in **Table 3.**

Table 3. Summary of Initial Guesses and Model Results for Five Variable Model

Guess Set	Initial Guess n_1 n_2 n_3 n_4 n_5	Best Fit of Round n_1 n_2 n_3 n_4 n_5	# of Gen	Perturbation Rate	Time Avg $(F_{err})_{mean}$	Min $(F_{err})_{mean}$
1	.000476 3.028 0.225 1.05 0.217	0.000516 2.920 .257 1.22 0.315	6000	9%	0.0390	0.0291
2	.000476 3.028 0.301 1.054 0.217	0.000536 2.81 0.356 1.54 0.296	6000	9%	0.03529	0.0248
3	.000537 2.81 0.356 1.54 0.296	0.000421 2.858 0.351 1.46 0.367	12000	12%	0.0346	0.0241
4	.000537 2.81 0.356 1.73 0.296	0.000382 2.80 0.364 1.69 0.505	12000	18%	0.0335	0.0229

For our initial guess we used the given values in the project document. Already we reached our goal of **Min $(F_{err})_{mean}$** less than 0.03 at .0291. To improve further we first tried changing n_3 from .225 to .301. This was because we noticed n_3 above the **$(g + n_4 g_{en} \gamma)$** term in **equation 1.5** in the project document and guessed it would highly influence our solution. As predicted, our **Min $(F_{err})_{mean}$** value successfully dropped to 0.0248. For our third round, seeing as the output best values for n_1-n_5 had shifted substantially after guess two, we tried inputting these exact values and increasing NGEN to 12000 and the perturbation to 12%. This is because we had moved to a new region in state space and wanted to "search the area" for any other local minima by increasing the nearby search radius. This only decreased our **Min $(F_{err})_{mean}$** value slightly to .0241. Finally, we further increased the perturbation to 18% and increased n_4 from 1.54 to 1.73 to further search the area and test the influence of n_4, also inside our **$(g + n_4 g_{en} \gamma)$** and guessed to have a reasonable impact on our solution.

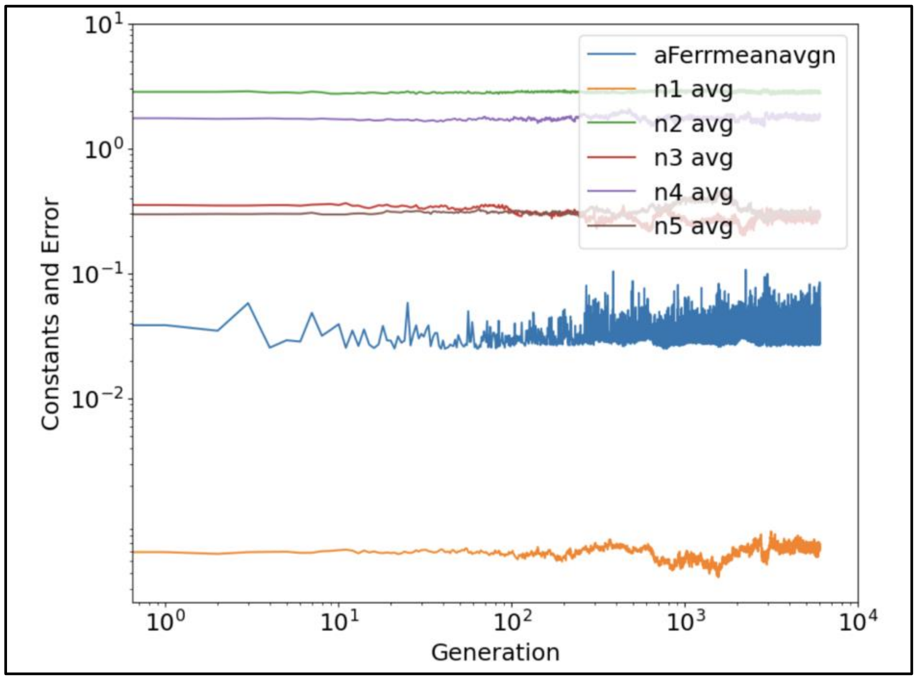


Figure 5. Variation in Constant Values during Best Fit Model Optimization for 5 Constant Model

Figure 5 shows the variation in values for each of the five *n* constants when the initial guesses were placed near our best fit solution. The n_2 constant remained extremely consistent as in the case of the three constant model. Since n_4 was relatively stable while n_1, n_3, and n_5 were not, it is likely n_2 and n_4 remained relatively constant even around similar solutions. At the best calculated values for n_1-n_5, our heat flux equation can be expressed as:

$$\mathbf{q'' = .00537(T_w - T_{sat})^{2.81} (g + 1.73g_{en}\gamma)^{.356} P^{.296}}$$

Both new terms added, namely n_4 and n_5 are by no means insignificant, signaling that our model found sizable correlations between *q"*, *P*, and γ and became noticeably more accurate compared with the three constant model. This indicates that pressure and surface tension parameter variation can deeply influence the heat flux. This effect can be seen in the **Figure 6** below in which the heat flux divided by the wall superheat $\boldsymbol{q''/(T_w - T_{sat})^{2.81}}$ is plotted against the surface tension parameter and gravitational constant:

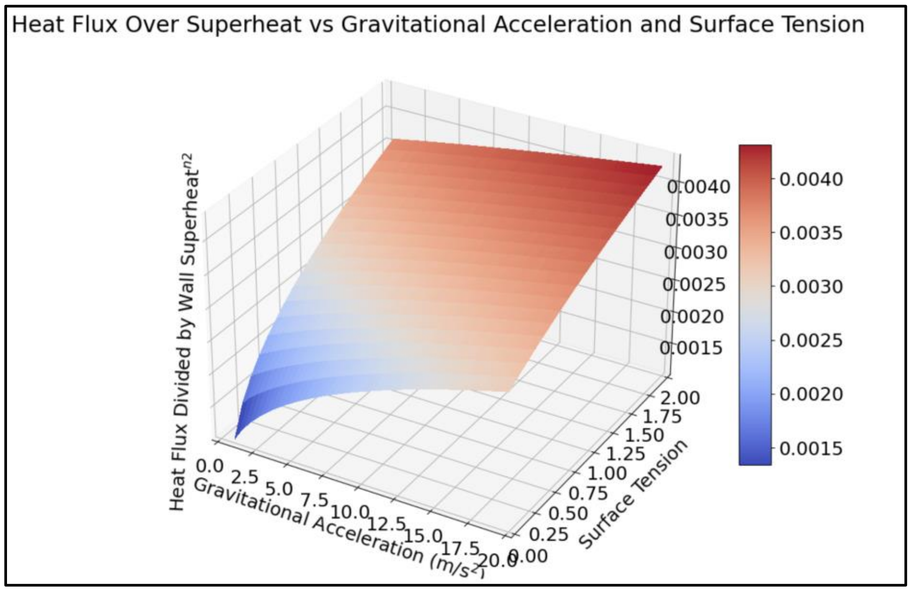


Figure 6. Surface Plot of Gravitational Acceleration, Surface Tension and Heat Transfer Efficiency

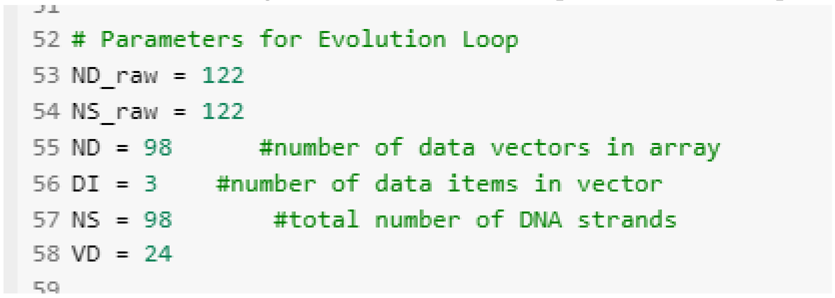
At larger surface tension and gravitational acceleration values, we see that the ratio between *q”* and $(T_w - T_{sat})^{2.81}$ increases and vice versa at lower *g* and *γ* values. $\boldsymbol{q''/(T_w - T_{sat})^{2.81}}$ can be thought of as efficiency (actually called the boiling heat transfer coefficient), which represents the efficiency of heat transfer during boiling. Surface tension parameter *γ* represents the percentage difference of the mixture’s surface tension compared to that of water. This means that as *γ* increases, the surface tension of our mixture moves further away from that of water. In this case, we would expect that larger *γ*’s would lead to higher heat transfer rates, and indeed we see this trend in the surface plot in the red areas. Furthermore, as *γ* nears 0, the solution experiences surface tension comparable to water, which we know is much harder to boil at lower *g*’s. Indeed we see again in the blue areas of the surface plot that at *γ* and *g* values closer to 0, the heat flux drops precipitously. From our solved equation for *q”* in the five constant model we also see that as *g* approaches 0, heat transfer efficiency should more rapidly approach 0, and indeed we see this in the surface plot as the slope rapidly decreases moving left toward g = 0. Moreover, as at high values of *g* (> 12 m/s^2), there is a linear relationship between surface tension and heat transfer efficiency. Yet, it is a nonlinear relationship when *g* is on the other end of that range.

Overall, when you consider *g* equating approximately 10 m/s^2 and the surface tension value of 0 to be the baseline for water boiling at sea-level, which is a blue-white color on **Figure 6**, it puts it into great perspective of the performance of a water/2-propanol mixture. As long as the surface tension parameter is greater than about 1.00, the mixture will be able to heat as good as water on earth in any gravitational acceleration.

4. Task 2 Analysis and Discussion

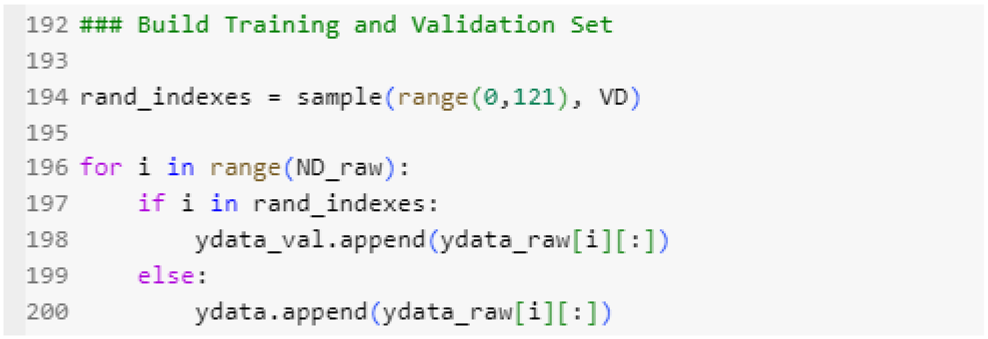
4.1. Code Modifications

1. First we Copy+Pasted our combined CodeP1.1F23 and CodeP1.2F23.
2. Next we updated the number of data vectors in the array. We created new constants ND_raw, NS_raw, and VD to the sizes for the full dataset and Validation dataset. We used ND, DI, etc. as the size for the training dataset. These were helpful to use for our plotting functions.



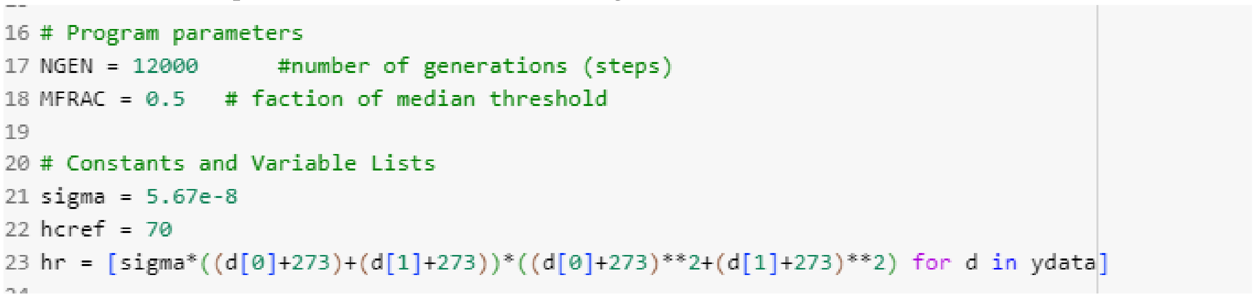
```
# Parameters for Evolution Loop
ND_raw = 122
NS_raw = 122
ND = 98      #number of data vectors in array
DI = 3    #number of data items in vector
NS = 98       #total number of DNA strands
VD = 24
```

3. We next wrote a few lines of code to randomly select training and validation datasets from the provided dataset.



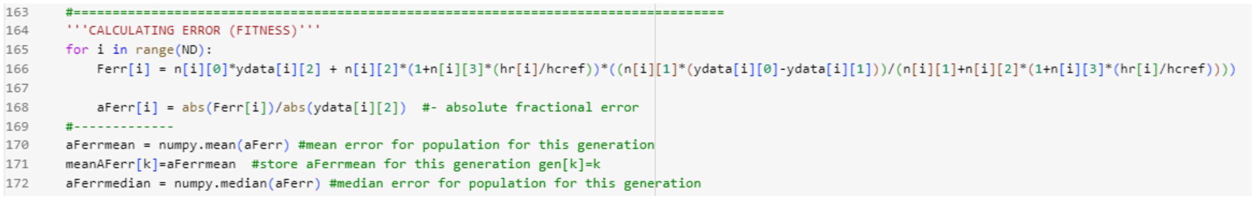
```
### Build Training and Validation Set

rand_indexes = sample(range(0,121), VD)

for i in range(ND_raw):
    if i in rand_indexes:
        ydata_val.append(ydata_raw[i][:])
    else:
        ydata.append(ydata_raw[i][:])
```

4. Next we varied the parameter NGEN and defined sigma, hcref, and hr.



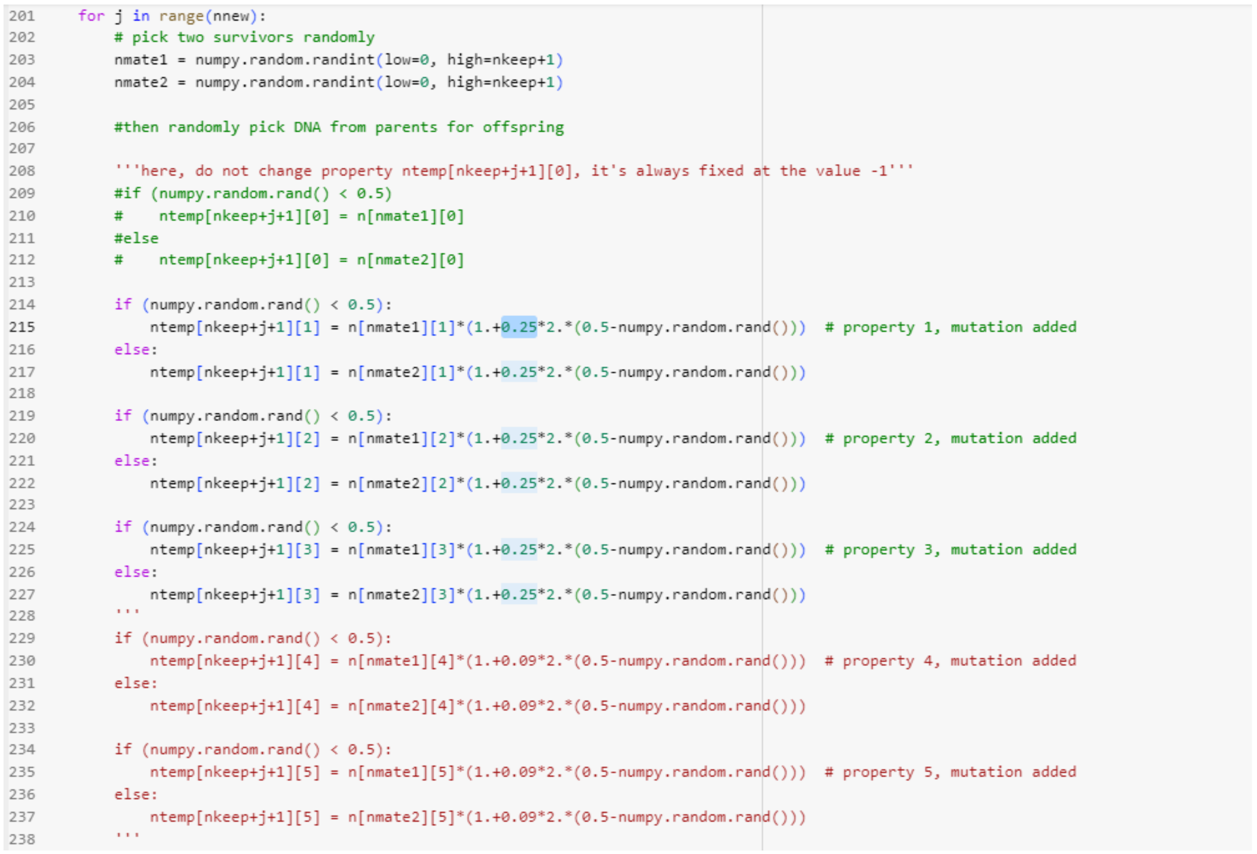
```
# Program parameters
NGEN = 12000      #number of generations (steps)
MFRAC = 0.5   # faction of median threshold

# Constants and Variable Lists
sigma = 5.67e-8
hcref = 70
hr = [sigma*((d[0]+273)+(d[1]+273))*((d[0]+273)**2+(d[1]+273)**2) for d in ydata]
```

5. Then we swapped in an updated error function for this task as defined in the project document in **equation 2.4**.



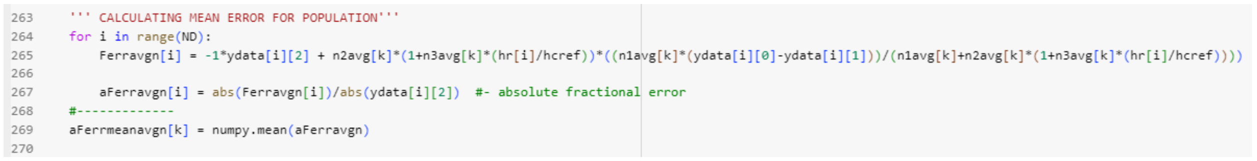
```
#=====================================================================================
'''CALCULATING ERROR (FITNESS)'''
for i in range(ND):
    Ferr[i] = n[i][0]*ydata[i][2] + n[i][2]*(1+n[i][3]*(hr[i]/hcref))*((n[i][1]*(ydata[i][0]-ydata[i][1]))/(n[i][1]+n[i][2]*(1+n[i][3]*(hr[i]/hcref))))

    aFerr[i] = abs(Ferr[i])/abs(ydata[i][2])  #- absolute fractional error
#-------------
aFerrmean = numpy.mean(aFerr) #mean error for population for this generation
meanAFerr[k]=aFerrmean  #store aFerrmean for this generation gen[k]=k
aFerrmedian = numpy.median(aFerr) #median error for population for this generation
```

6. We next played with the perturbation values for mutation. See highlighted values below:



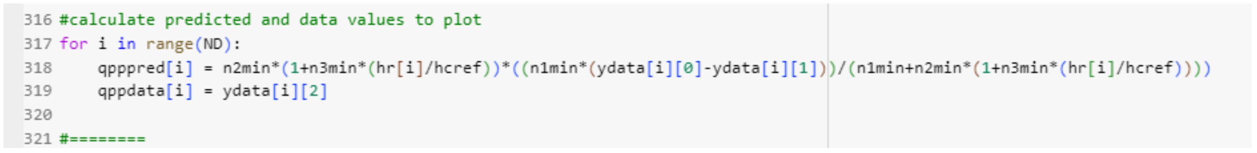
```
 for j in range(nnew):
     # pick two survivors randomly
     nmate1 = numpy.random.randint(low=0, high=nkeep+1)
     nmate2 = numpy.random.randint(low=0, high=nkeep+1)

     #then randomly pick DNA from parents for offspring

     '''here, do not change property ntemp[nkeep+j+1][0], it's always fixed at the value -1'''
     #if (numpy.random.rand() < 0.5)
     #    ntemp[nkeep+j+1][0] = n[nmate1][0]
     #else
     #    ntemp[nkeep+j+1][0] = n[nmate2][0]

     if (numpy.random.rand() < 0.5):
         ntemp[nkeep+j+1][1] = n[nmate1][1]*(1.+0.25*2.*(0.5-numpy.random.rand()))  # property 1, mutation added
     else:
         ntemp[nkeep+j+1][1] = n[nmate2][1]*(1.+0.25*2.*(0.5-numpy.random.rand()))

     if (numpy.random.rand() < 0.5):
         ntemp[nkeep+j+1][2] = n[nmate1][2]*(1.+0.25*2.*(0.5-numpy.random.rand()))  # property 2, mutation added
     else:
         ntemp[nkeep+j+1][2] = n[nmate2][2]*(1.+0.25*2.*(0.5-numpy.random.rand()))

     if (numpy.random.rand() < 0.5):
         ntemp[nkeep+j+1][3] = n[nmate1][3]*(1.+0.25*2.*(0.5-numpy.random.rand()))  # property 3, mutation added
     else:
         ntemp[nkeep+j+1][3] = n[nmate2][3]*(1.+0.25*2.*(0.5-numpy.random.rand()))
     '''
     if (numpy.random.rand() < 0.5):
         ntemp[nkeep+j+1][4] = n[nmate1][4]*(1.+0.09*2.*(0.5-numpy.random.rand()))  # property 4, mutation added
     else:
         ntemp[nkeep+j+1][4] = n[nmate2][4]*(1.+0.09*2.*(0.5-numpy.random.rand()))

     if (numpy.random.rand() < 0.5):
         ntemp[nkeep+j+1][5] = n[nmate1][5]*(1.+0.09*2.*(0.5-numpy.random.rand()))  # property 5, mutation added
     else:
         ntemp[nkeep+j+1][5] = n[nmate2][5]*(1.+0.09*2.*(0.5-numpy.random.rand()))
     '''
```

7. Next we updated the fitness calculation for the population at the end of the loop, again utilizing **equation 2.4** in the project document.



```
''' CALCULATING MEAN ERROR FOR POPULATION'''
for i in range(ND):
    Ferravgn[i] = -1*ydata[i][2] + n2avg[k]*(1+n3avg[k]*(hr[i]/hcref))*((n1avg[k]*(ydata[i][0]-ydata[i][1]))/(n1avg[k]+n2avg[k]*(1+n3avg[k]*(hr[i]/hcref))))

    aFerravgn[i] = abs(Ferravgn[i])/abs(ydata[i][2])  #- absolute fractional error
#-------------
aFerrmeanavgn[k] = numpy.mean(aFerravgn)

```

8. Next we updated the function used for calculating predicted heat flux using **equation 2.6** in the project document.



```
#calculate predicted and data values to plot
for i in range(ND):
    qpppred[i] = n2min*(1+n3min*(hr[i]/hcref))*((n1min*(ydata[i][0]-ydata[i][1]))/(n1min+n2min*(1+n3min*(hr[i]/hcref))))
    qppdata[i] = ydata[i][2]

#========
```

4.2. Analysis and Discussion

For Task 2, we continued our exploration of modeling heat transfer performance across various parameters, specifically for heat pipes. The conduction and radiation performance of the finned structures, wick structures, and condensers of heat pipes can be described by thermal conductance rates and radiation correction values specific to their design. Traditionally, these values can be computed and estimated theoretically and plugged into a model to predict heat transfer rates. Alternatively, we use machine learning tools to iterate through a model and find a set of thermal conductance rates and a radiation correction value that best describes the heat pipe based on experimentally collected data.

A simple model for a heat pipe's performance (heat flux q'') can be constructed as a function of the internal air temperature ($T_{a,in}$) and external air temperature ($T_{a,out}$) with σ and $h_{c,ref}$ being constants and h_r being a function of $T_{a,in}$ and $T_{a,out}$:

$$q'' = n_2 (1 + n_3 (h_r / h_{c,ref})) [(n_1 (T_{a,in} - T_{a,out})) / (n_1 + n_2 (1 + n_3 (h_r / h_{c,ref})))]$$

Specifically, we aimed for the genetic algorithm to optimize the constants: 1) n_1, the thermal conductance of the evaporator fin section 2) n_2, the thermal conductance of the condenser fin section, and 3) n_3 the radiation correction term, that would best suit the model. Similar to Task 1, our fitness measurement for each generation equated to the mean of our total error function $(F_{err})_{\text{mean}}$, which calculates the absolute fractional error of each data point. We were attempting to minimize $(F_{err})_{\text{mean}}$ below 0.05. While running the algorithm, we explored five different sets of initial guesses as shown in **Table 4**. Note that we trained the model solely on the training dataset, which composed 80% of the full dataset. The other 20% of the dataset, the validation dataset, was saved for testing the model.

Table 4. Summary of Initial Guesses and Model Results for Three Variable Model

Guess Set	Initial Guess n_1 n_2 n_3	Best Fit of Round n_1 n_2 n_3	# of Gen	Perturbation Rate	Time Avg $(F_{err})_{mean}$	Min $(F_{err})_{mean}$
1	6.00 6.00 250	3.07 2.71 198	6000	20%	0.0298	0.017707
2	6.00 6.00 250	3.07 3.54 153	12000	20%	0.0361	0.017705
3	3.00 3.00 100	3.21 2.45 126	12000	25%	0.0437	0.017691
4	3.30 3.50 70.0	3.33 4.74 41.1	12000	15%	0.0262	0.017716
5	3.30 3.50 150	3.15 2.49 147	12000	15%	0.0245	0.017700

By manipulating the perturbation rate for mutation, the number of generations, and the *n* constant guesses we gained better knowledge about the behavior of the model. Overall, the algorithm performed well at converging to minimum solutions. Across all five guess sets, the minimum fractional errors were all below 0.0178, a value our model in Task 1 did not come close too.

Similar to Task 1, we noticed that the algorithm experienced greater optimization success when we used higher perturbation rates for mutation. However, the difference in minimum fractional error for each set was fairly small, making it more difficult to discern if the perturbation rate or the strong performance of the algorithm was the main contributor to its success. Additionally, we examined the effect of the NGEN variable by only altering the number of generations in the first two guesses. With both 6000 and 12000 generations, the algorithm performed surprisingly well. Both times, the model was optimized and had a minimum fractional error of around 0.01771. It is important to note however that if the initial guesses were not as close to the final solution (e.g. all around 60), the number of generations would have more of an impact on the model's success.

The most surprising detail of our optimization was the consistency of n_1. In each of our guess sets, n_1 converged to approximately 3. Conversely, n_3 varied drastically across a large range of 41 to 198. The impact of the radiation correction term n_3 was likely negated by the term $(h_r / h_{c,ref})$, which on average was less than 0.075. So n_3 held less weight. We also observed that n_1 had a strong influence on the fitness of the model and allowed the algorithm to converge quickly, in some cases after only 20 generations. For example, with an initial guess of 6 for n_1, the fitness improved dramatically as soon as the n_1 value converged around 3 as shown in **Figure 7**. Of all the constants, n_1 had the largest impact on model optimization performance and speed to convergence.

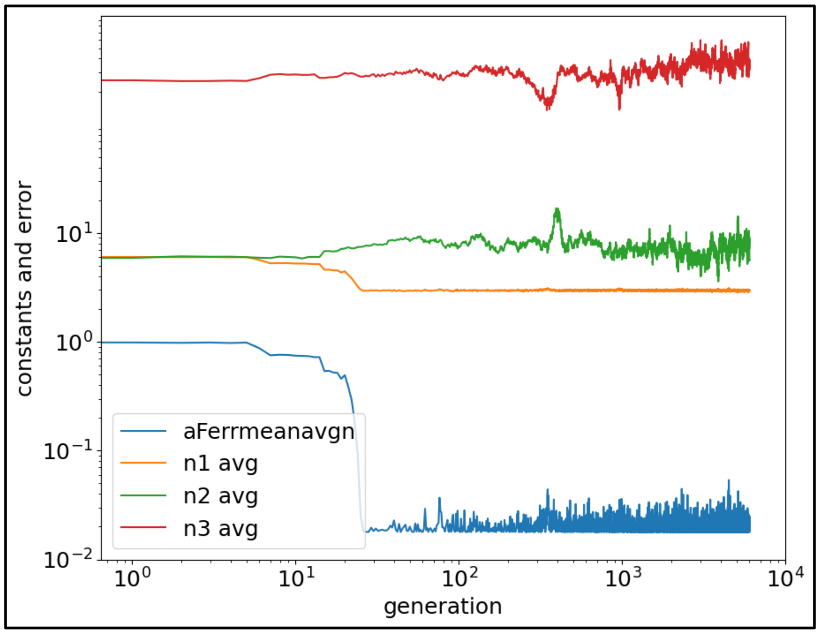


Figure 7. Variation in Constant Values during Guess Set 1 Optimization

After several rounds of experimenting, we converged on our best fit solution shown in the equation below:

$$q'' = 3.66\,(1 + 88.6\,(h_r / h_{c,ref}))\ [\ (3.18\,(T_{a,in} - T_{a,out}))\ /\ (3.18 + 3.66\,(1 + 88.6\,(h_r / h_{c,ref})))\ ]$$

To get a closer look at the performance of the model, we ran the model with data from both the training and validation dataset. Overall, the model performed extremely well, even on both datasets. Examining **Figure 8**, the regression line, fitted to the entire dataset, shows a strong linear relationship between the predicted and measured values. In fact, the fitted line's slope of 1 indicates that the model predicts heat flux values for the heat pipe that closely resemble the measured values. This even is occurring with logged values, which tends to exaggerate outliers.

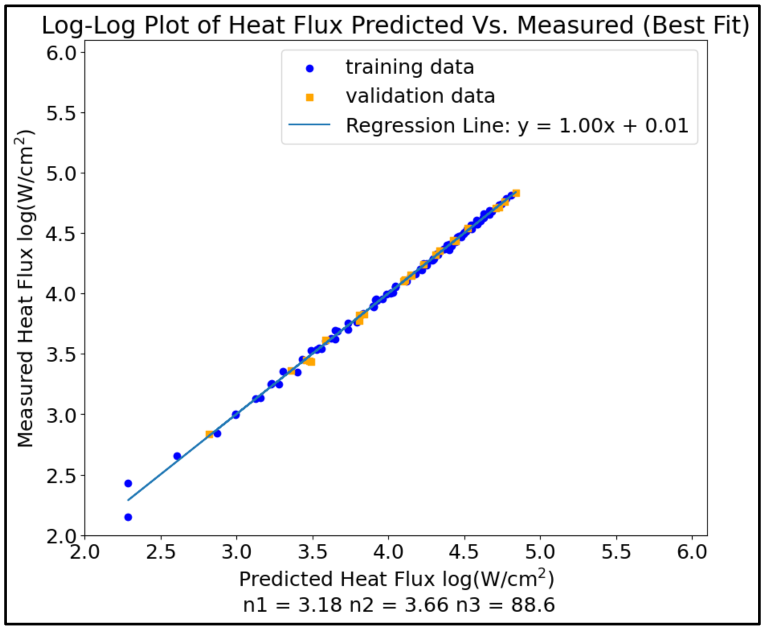


Figure 8. Log-Log Plot of the Heat Flux Predicted vs Measured (Best Fit)

Furthermore, in **Table 5** we highlight the difference in fitness test of the training vs validation data.

Table 5. Performance of Training and Validation Data of Heat Pipe Model

	Training Data Min(F_{err})$_{mean}$	ValidationData Min(F_{err})$_{mean}$
Heat Pipe Model	0.01769	0.01537

When tested on new, unseen validation data, the model achieved a slightly lower minimum mean error. This indicates reasonable generalization capability for the model. It also tells us that the model doesn't simply "memorize" the training data and overfit, but is able to perform across new terrain, indicating higher model quality. In both the training and validation cases, the errors are relatively low, indicating that the model's predictions are generally close to the actual values. The data also helps us understand key relationships between internal and external temperature, and the heat transfer rate as displayed in **Figure 9** below.

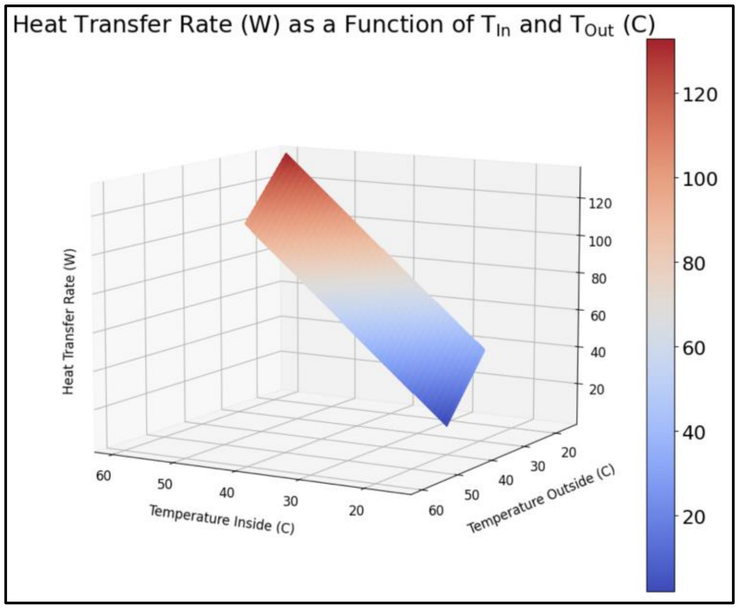


Figure 9. Surface Plot of Heat Transfer Rate vs Temperatures

From our data, we know that the inside temperature ranges roughly from 27-60˚C while the outside temperature ranges roughly from 16-27˚C. We also know from the laws of thermodynamics that the efficiency of heat transfer increases with a larger difference in temperature between our cold and hot reservoirs (one might think of Carnot's model of an engine which states that we can relate efficiency to T_H-T_C/T_H where T_H is the temperature of a hot reservoir and T_C is the temperature of a nearby cold reservoir – in this case, the inside and outside temperatures of our heat pipe, respectively). The surface plot is as expected with a higher heat transfer rate at larger inside temperature values. This is sufficient as the outside temperature values remained relatively low and constant compared to the inside temperature values. Similarly we see that at any constant inside temperature value, the efficiency increases slightly as outside temperature drops (this may be difficult to see in the figure displayed, however the plane slopes upward as you move along the right side of the outside temperature axis). This further reflects an increase in heat transfer rate, or efficiency, as the temperature difference increases.

5. Discussion of Raw versus Dimensionless Data Analysis

Throughout our data analysis for this project, we were able to explore the different methods that can be used when analyzing relationships in nature. For both tasks, we attempted to make sense of relationships between parameters and overall performance by taking into account a mix of raw and dimensionless data, each of which provide their own advantages and disadvantages.

In essence, raw data refers to the original, unprocessed data collected in experiments. They contain units of measurement like temperature, length, and etc. On the other hand, dimensionless data is data that has been transformed to be unitless. In other words, it is usually raw data that has been divided by a reference value of the same unit. It can be thought of as a ratio between two numbers or measurements.

The main advantage of analyzing scientific relationships with raw data is that it provides you with a contextual understanding. When you look at the data, you have a clear understanding of what the experiment is measuring. More importantly, you can see how input experimental conditions affect the output result in its actual units. For example, for Task 2, we were able to clearly see that when the temperature difference between the interior and exterior increased, the heat transfer rate or Watts increased. Thus, its simplicity makes it easier for non-scientific people to understand what is occuring. However, the strong disadvantage of analyzing raw data is that it has a limited generalization. It often does not provide enough context to analyze fundamentally what is occurring. Raw data tells you how hot or cold something is, but it doesn't directly tell you the relationship of temperature with another variable. Thus, it is challenging to generalize findings across different experiments and conditions.

In contrast, dimensionless data analysis takes full advantage of the area that raw data analysis struggles in: it can encapsulate the fundamental science of what is occurring, allowing the findings to be generalized. Often they represent ratios between several variables that can provide direct insight into the relationships being analyzed in an experiment. For instance, by merging heat flux and wall superheat into a dimensionless data point for our surface plot in Task 1, we created a unitless variable that represented overall heat transfer efficiency. We then better understood how gravitational acceleration and surface tension affect it. Thus, dimensionless data analysis makes it easier to identify trends and patterns, which can identify universal behaviors that can be applied elsewhere. However, they do suffer from a loss of specificity. If you are hoping to analyze one variable in particular, then focusing on raw data would be more advantageous. They also are not very intuitive at first glance as they can be complicated. Thus, they will be difficult for non-scientific people to understand quickly. The choice between raw data analysis and dimensionless data analysis hinges on research objectives, as each methodology brings unique strengths to the table.

6. Conclusion

In Task 1, we investigated boiling heat transfer in water/2-propanol mixtures under different gravitational conditions. Our model revealed that heat flux is significantly influenced by wall superheat, with gravity playing a subtle yet discernible role. The incorporation of additional parameters, such as pressure and surface tension, expanded our model's accuracy, emphasizing the importance of these factors in heat transfer efficiency.

Task 2 delved into the performance of heat pipes, employing an algorithm to optimize thermal conductance rates and a radiation correction value. Our findings underscored the pivotal role of n_1, the thermal conductance of the evaporator fin section, in improving the model optimization performance. In the end, we achieved a best-fit solution that provides precise predictions of the heat transfer rates of heat pipes.

Furthermore, our project highlighted the significance of analyzing data in both raw and dimensionless forms. While raw data offers valuable contextual insights by including units and shedding light on individual variable values, dimensionless data unravels fundamental relationships in the data.

Finally, this project provided valuable insights into the complex world of heat transfer and the machine learning tools that can help model it. Not only do models help accurately predict an output given specific input variables, they also paint a clearer picture of how they are all related. Moreover, with strong knowledge about the behaviors of setting initial guesses, perturbation rates, and number of generations, one can further push the envelope of how quick, accurate, and refined a model can behave.