Mechanical Engineering Department — ME249, Fall 2023
UC Berkeley — Prof. V. Carey

Project 2. **Due date: October 26, 2023**

Part 2.
Code files used in Part 2: CodeP2.3F23.ipynb, CodeP2.4F23.ipynb, CodeP2.5F23.ipynb

General Information

Part 2 of this project deals with analysis of the hybrid solar fossil-fuel gas turbine system in the figures below.

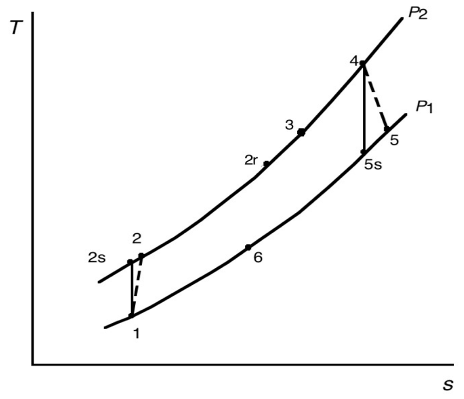


Figure 1.

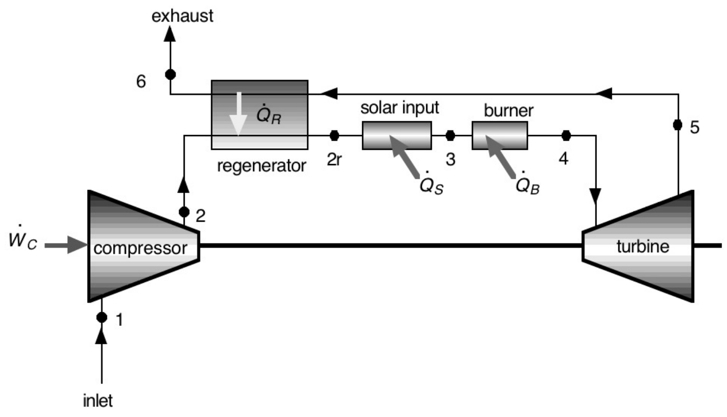


Figure 2.

In this system, air at atmospheric pressure $P_1 = 101$ kPa and at a temperature T_1 enters the compressor inlet at a flow rate typically about 6.0 kg/s. The air flowing out of the compressor at the high-side pressure first is heated to temperature T_{2r} in the regenerator, transferring waste heat from the turbine exhaust stream. The gas is then heated further in an exchanger that delivers solar thermal heat input, raising the temperature to T_3. Finally, the air flows into a burner where fuel is injected and burned to raise the temperature to T_4. Having T_4 as high as possible is thermodynamically advantageous, resulting in higher system efficiency for higher T_4 . However, if T_4 is too high, the components of the turbine may be damaged, and for that reason, an optimal T_4 temperature is usually specified which is the highest value that can be tolerated by the blade materials in the turbine. Recent development efforts have resulting in turbine designs, with special blade materials, that can operate at 1600 K. Here the system is specified to operate at 1473 K (1200 °C), with a pressure ratio P_2 / P_1 of 14.

The burner in the system is designed to burn pure methane or a mixture of methane (CH_4) with some added propane (C_3H_8). The mixture ratio may be dictated by cost factors and availability, resulting in the mole fraction of the fuels ranging from 0% propane and all methane, to 50% propane and 50% methane, by mole. Key parameters here are the molar air-to-fuel ratio α and the fuel propane mole fraction γ defined as.

α = (moles of air)/(total moles of propane and methane in mixture) in inlet flow to burner
γ = (moles of propane)/(moles of propane and methane in fuel mixture)

A *stoichiometric* mixture has just enough oxygen in the air to convert all the propane and methane to H_2O and CO_2, with no oxygen left over. For arbitrary γ and a value of α greater than or equal to the stoichiometric value, the balanced chemical reaction can be written as

$$\left(\frac{\alpha}{4.76}\right)(O_2 + 3.76\ N_2) + [\ \gamma C_3H_8 + (1-\gamma)CH_4\ ] \dashrightarrow \left(\frac{3.76\alpha}{4.76}\right)N_2 + \left(\frac{\alpha}{4.76} - 3\gamma - 2\right)O_2$$

$$+ (2+2\gamma)\ H_2O + (1+2\gamma)CO_2$$

Note that the above equation scales the number of moles of other species on a per-mole-of-fuel-mixture basis. For stochiometric conditions,

$$\alpha = \ \alpha_{stoich} = 4.76\,(2+3\gamma)$$

The burner operates adiabatically, and the hottest exit temperature corresponds to stoichiometric conditions. For $\alpha > \alpha_{stoich}$, the exit temperature will be lower, and therefore, adding excess air (raising α) is a way of controlling the burner exit temperature.

Air inlet temperature T_1, solar heat input $\dot{Q}_S$, and the fuel mixture fraction vary during system operation, and consequently, the rate of fuel injection in the burner must be varied to hold the turbine inlet temperature T_4 at the target level of 1473 K. To facilitate model-based control of T_4, a model that predicts the required air-to fuel ratio to achieve this turbine inlet temperature under varying conditions can be used by a controller to set the fuel flow rate at the proper values. Two ways of constructing a model are:

1. Construct a physical model based on thermodynamics, fluid dynamics and heat transfer.

2. Run performance tests for the system over the expected range of operating conditions and construct a mathematical/computational fit to the experimental data using machine learning methods.

Option 1, construction of a physical model, has advantages and drawbacks. Use of a theoretical model can avoid the need for building a prototype system and obtaining test data. However, physical models generally incorporate idealizations that may not be completely accurate, and they may require knowledge of system parameters that may not be known with complete accuracy. For example, the regenerative heat exchanger may be modeled with the assumption that its heat transfer conductance is uniform throughout the unit, which may not be accurate. Also, the value of the conductance may not be known precisely. Other parameters such as the compressor isentropic efficiency and the turbine isentropic efficiency also may vary with conditions, and not be known to high accuracy. Creating an accurate model this way may be challenging.

Option 2 requires obtaining test data. However, creating a machine learning model to fit the multivariate data accounts for all the real system parametric effects and requires no assumptions or idealizations. If a good fit to the data is obtained, the predictions of the resulting model can be accurate to the level of the uncertainty in the data and the mean absolute error or RMS error of the fit. In addition, data obtained during field operation of the systems can be used to update the model accounting for variation in its performance during its operational lifetime.

The objective of Part 2 of this project is to construct a neural network model of the performance of the gas turbine system shown in Figures 1 and 2 above. The physics of the system operation indicates that the parameters that would have to be specified to determine the performance at a given operating point from a physical model are:

pressures P_1, P_2
temperatures T_1, T_4
the compressor isentropic efficiency, η_c
the turbine isentropic efficiency, η_t
the regenerator effectiveness, ε_r
the mass flow rate of air into the compressor $\dot{m}_{air}$
the rate of solar thermal heat input, $\dot{Q}_S$
(moles of air)/(moles of propane and methane mixture) in the burner, α
(moles of propane)/(moles of propane and methane in fuel mixture) in the burner, γ

the predicted performance can be quantified as:

$$\text{the system efficiency} = \eta_{sys} = \frac{\text{(turbine power out) - (compressor power in)}}{\text{combustion} + \text{solar heat input rates}}$$

Here it is assumed that the system design fixes $P_1 = 101$ kPa, $P_2 = 14{*}101$ kPa, η_c, η_t, ε_r, and that the control scheme will adjust the air to fuel ratio to keep $T_4 = 1473$ K. This target value is presumed to be fixed. The air flow rate into the compressor, $\dot{m}_{air}$, is assumed to be dictated by the design compressor speed that results in 6 kg/s at 298K, but varies significantly with inlet temperature (that affects air density) which may range from -5°C to 45°C.

The result is that for this system, the required air to fuel ratio α and the efficiency of the system η_{sys} are dictated by operating conditions that are primarily specified by values of T_1, γ and $\dot{Q}_S$:

$$[T_1, \gamma, \dot{Q}_S] \quad \Leftrightarrow \quad [\alpha, \eta_{sys}]$$

Task 2.1
The accompanying python code file CodeP2.3F23.ipynb contains two arrays `xdata` and `ydata` that contain the input (operating condition) parameters $[T_1, \gamma, \dot{Q}_S]$ and output parameters $[\alpha, \eta_{sys}]$, respectively, for the system of interest. In the first cell of the skeleton code in the provided file CodeP2.4F23.ipynb, make the following modifications:

(i) Extend the five data point version of `xdata` and `ydata` to include all the data points in the CodeP2.3F23.ipynb file. To check they have been input correctly, print out the arrays in dimensional form before the normalizing them in the next step.
(ii) Determine the median value for each of the parameters in `xdata` and `ydata` (T_1, γ, $\dot{Q}_S$, α, and η_{sys}) (values in the code are placeholders). Then, modify each array, replacing the raw data value by the raw data value divided by its respective median value. When this is complete, the new `xdata` and `ydata` arrays should contain data that has been normalized with the median value for that variable. As a final step, create print statements that print the normalized `xdata` and `ydata` arrays when cell 1 of the program is run.

Task 2.2
(a) In the second cell of the code in file CodeP2.4F23.ipynb, create a sequential neural network with three input variables (`input_shape=[3]`)in the first (hidden) dense layer of the network list. Initially set the number of neurons to 4 in this layer and set the activation function to 'K.relu.'

Then add two more hidden dense layers: the second hidden layer of the network with 8 neurons and the third hidden dense layer with 4 neurons. In both of these layers, set the activation function to 'K.relu'. Finally, add one more output dense layer with 2 neurons (one for each output) and do not specify an activation function for the output layer.

When you are finished, the list of layers in your sequential network should have 4, 8, 4, and 2 neurons. In cell 2, set the initializer limits to minval= -0.9, maxval=0.9. Run the first and second cells of the modified CodeP2.4F23.ipynb code. You should get no error messages after the second cell if things are in order.

(b) In the third cell, just change the learning parameter to 0.001, and run that cell to check that no error message appears. In the fourth cell, set epochs equal to 600. Then run the first four cells in sequence. Note that this network should be trained using the normalized data output from Task 2.1. The fourth cell will train the network by successive forward passes for each point in the data set, and a backpropagation pass to update the weights and biases at the end of each epoch. For each epoch, the reported loss value indicates the mean absolute error. Since our normalized parameters make the data values of order 1, we want to achieve a value of mean absolute error that is small (a few percent) of one (~0.05 or less).

Note you can run cell 4 repeated times, and each time it will begin with the constants from the last epoch in the previous run, so this is a way to extending the number of epochs to try to further reduce the mean absolute error. If you go back and start with cell 1, it starts over completely. Run cell 4 repeated times and try to get as low a loss value as possible (below 0.04 if possible). Report the resulting lowest loss value in your written summary.

Cell 5 in the CodeP2.4F23.ipynb code provides an example of how to predict values of α, and η_{sys} for a submitted set of the input variables T_1, γ, $\dot{Q}_S$. You can put this in a loop to compute predicted values of α, and η_{sys} for whatever input you want. This cell also creates a plot of the dimensionless data value versus the precited value of nondimensional α. Fix γ equal to 0.25 and use the model to predict the variation of α for 268 < T_1 < 318 K and 500 < $\dot{Q}_S$ < 2500 kW. Use the results to create a surface plot for α as a function of T_1 and $\dot{Q}_S$ over these ranges.

Task 2.3

(a) As noted above, cell 5 in the CodeP2.4F23.ipynb code indicates how to predict values of α, and η_{sys} for a submitted set of the input variables T_1, γ, $\dot{Q}_S$. Below is a table of test data (distinct from the training data). Note that you have to normalize the input data using the same median values as the trained model, insert the normalized values into an array, and submit the array to the `model.predict` function. The output results must be multiplied by the median values using in training to get physical output values.

Test Data

$[T_1, \gamma, \dot{Q}_S]$		$[\alpha, \eta_{sys}]$
[318.0 , 0.0 , 500.0]		[35.13 , 0.3808]
[318.0 , 0.0 , 1500.0]		[47.46 , 0.3930]
[318.0 , 0.0 , 2500.0]		[73.12 , 0.4061]
[318.0 , 0.25 , 1500.0]		[66.34 , 0.4098]
[318.0 , 0.5 , 500.0]		[63.09, 0.4154]
[318.0 , 0.5 , 1500.0]		[85.23 , 0.4197]
[318.0 , 0.5 , 2500.0]		[131.32 , 0.4242]
[303.0 , 0.0 , 1000.0]		[38.99 , 0.4012]
[303.0 , 0.0 , 2000.0]		[53.80 , 0.4136]
[303.0 , 0.25 , 1000.0]		[54.51 , 0.4215]
[303.0 , 0.25 , 2000.0]		[75.22 , 0.4290]
[303.0 , 0.5 , 1000.0]		[70.04, 0.4337]
[303.0 , 0.5 , 2000.0]		[96.65, 0.4382]
[288.0 , 0.0 , 500.0]		[33.45 , 0.4091]
[288.0 , 0.0 , 2500.0]		[60.80 , 0.4334]
[288.0 , 0.25 , 2500.0]		[85.044, 0.4477]
[288.0 , 0.5 , 1500.0]		[77.56 , 0.4516]
[268.0 , 0.0 , 1500.0]		[40.68 , 0.4383]
[268.0 , 0.25 , 2000.0]		[65.24 , 0.4628]
[268.0 , 0.5 , 2500.0]		[98.23 , 0.4760]

Plot α for the test data versus the corresponding predicted α values on a log-log plot to evaluate the agreement. Also determined the rms deviation between the predictions and this collection of test data.

Task 2.4

The accompanying python code file CodeP2.5F23.ipynb also contains a skeleton program to train a neural network to predict the system output parameters $[\alpha, \eta_{sys}]$ for the input operating condition parameters $[T_1, \gamma, \dot{Q}_S]$. This code CodeP2.5F23.ipynb differs from CodeP2.4F23.ipynb in that the weights and bias values for optimal fit are determined using a genetic algorithm. To do so, the machinery that facilitates the backpropagation calculations in CodeP2.4F23.ipynb is replaced by code that minimizes the loss function using a genetic algorithm. This code makes use of a special python code package: pygad which is not part of the usual standard python software installation.

(a) The pygad package needs to be installed, and tips on how to do so are indicated below:

If you are using a custom environment in anaconda jupyter notebook:

(i) Open the Environments tab in Anaconda, click your custom environment (for 3.7), click the arrow next to your custom environment, and select Terminal from the drop down menu. In the Terminal window, at the prompt, enter the command:

```
>> pip install pygad
```

And hit the enter key. This should produce output statements indicating that pygad has been successfully installed.

(ii) Open a new notebook file and in the first code cell of the notebook, type the following commands:

```
!pip install pygad
'''To use pygad, first import it.'''
import tensorflow.keras
import numpy
import pygad
print('pygad installed correctly if no error messages')
```

Run the cell containing the commands typed in step (i) above. This should produce output statements indicating that pygad has been successfully installed. If this installation is successful, you should be able to install the code in file CodeP2.5F23 in cell 2, or cells 2 and 3 if you want to run it in segments That code will be executed by running cells 2, or 2 followed by 3.

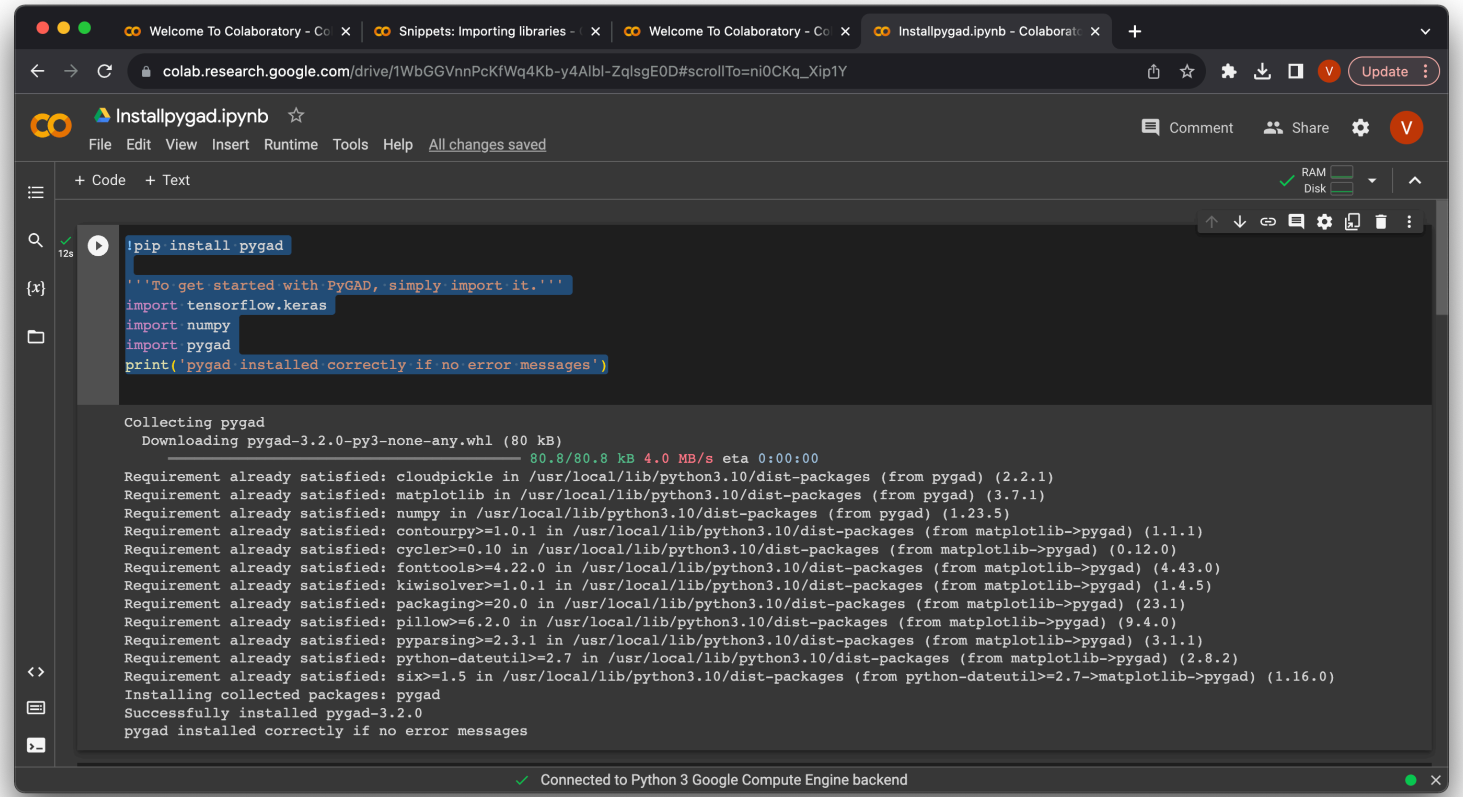
If you are using the Colab.com web site:

(i) Open up a new notebook at the Colab web site. In the first code cell of the notebook, type the following commands:

```
!pip install pygad

'''To use pygad, first import it.'''
import tensorflow.keras
import numpy
import pygad
print('pygad installed correctly if no error messages')
```

(ii) Run the cell containing the commands typed in step (i) above. This should produce output statements indicating that pygad has been successfully installed. If this installation is successful, you should be able to install the code in file CodeP2.5F23 in cell 2, or cells 2 and 3 if you want to run it in segments That code will be executed by running cells 2, or 2 followed by 3.


Welcome To Colaboratory - Co
Snippets: Importing libraries -
Welcome To Colaboratory - Co
Installpygad.ipynb - Colaborat
colab.research.google.com/drive/1WbGGVnnPcKfWq4Kb-y4AlbI-ZqlsgE0D#scrollTo=ni0CKq_Xip1Y
Update
Installpygad.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
12s

```
!pip install pygad

'''To get started with PyGAD, simply import it.'''
import tensorflow.keras
import numpy
import pygad
print('pygad installed correctly if no error messages')
```

```
Collecting pygad
  Downloading pygad-3.2.0-py3-none-any.whl (80 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 80.8/80.8 kB 4.0 MB/s eta 0:00:00
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from pygad) (2.2.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from pygad) (3.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pygad) (1.23.5)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pygad) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pygad) (0.12.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pygad) (4.43.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pygad) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pygad) (23.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pygad) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pygad) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pygad) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->pygad) (1.16.0)
Installing collected packages: pygad
Successfully installed pygad-3.2.0
pygad installed correctly if no error messages
```

Connected to Python 3 Google Compute Engine backend

Figure 3.

(b) Once pygad is installed and you have copied the contents of file CodeP2.5F23 into one or two subsequent cells of a notebook, make the following changes to this program:

(i) Install the full data set (normalized with median values) that you used in CodeP2.4F23.ipynb in Tasks 2.1 and 2.2 into the data prep section of CodeP2.5F23.ipynb.

(ii) in the initializer, set minval = -0.9 and maxval = 0.9
(iii) set num_generations = 1500

For all the cases to be tested, leave these parameters unchanged:
num_solutions=40
num_parents_mating = 7
sol_per_pop = 37 (batch size)
parent_selection_type = "sss" (steady state replacement)
keep_parents = 5 (number kept per generation)
crossover_type = "single_point"
mutation_type = "random"
mutation_percent_genes = "default" (= 10%)

When you have made the changes specified above you will have set up case 1 in the table below. Consider this to be the baseline case.

Case	1	2	3	4	5	6
NN layers	4,8,4,2	4,8,4,2	4,8,4,2	4,8,4,2	4,5,4,3,2	4,5,4,3,2
Act. Funct.	relu	elu	relu	elu	relu	elu
w,b minval	-0.9	-0.9	-2.9	-0.4	-2.9	-0.9
w,b maxval	0.9	0.9	2.9	0.4	2.9	0.9
Num_solutions	40	40	40	40	40	40
Number of generations (initial)*	1500	1500	1500	1500	1500	1500
Best Fitness value at end:						
Corresponding best combined mae:						

*Larger values may be used if that improves the fit to the data

Run the program for the baseline case a few times to get the best fit possible for that case. You can run it for more generations than the indicated 1500 if you think that will significantly improve the fit. Enter your best fit into the summary table above, and save the best fit plot of dimensionless α for the data versus the best-fit predicted dimensionless α for inclusion in your report.

(c) After finding a best fit for case1, run the code several times for each of the other cases in the table. For your report, document the best fit found for each case, indicating the fitness and mae of the best fit in the summary table, and a plot of the dimensionless data α versus the best-fit predicted dimensionless α for each case.

Finally, identify the best fit of all the 6 cases you considered, and compare its results to those for the best fit obtained with backpropagation in Task 2.2. Based on your comparisons, in your report, assess the relative effectiveness backpropagation or a genetic algorithm for training a neural network. Is the effectiveness of the two they about the same, or does it seem that one is preferable in this case. To the extent possible, justify your arguments with your computed results.

Project 2 Tasks to be divided between coworkers:

(1) Program modifications for Part 1
(2) Interpretations of results for Part 1
(3) Data prep for Part2

(3) Program modifications for neural network modeling in Part 2
(5) Plotting and analysis of the results for Part 2
(6) Write-up of the results and conclusions

Deliverables:
Written final report should include:

(1) Written summary of how the work was divided between coworkers.

(2) An assessment of the results comparisons for the first principles program and the keras neural network program

(3) An assessment of the impact of neural network design variations explored in Part 2.

(4) An assessment of comparisons of the results for the back propagation trained neural network with those for the genetic algorithm trained neural network in Part 2.
(5) Plots requested in Parts 2
(6) A copy of the each of the codes your created for this project should be include in an appendix for your report.

Grade will be based on:
(1) thoroughness of documentation of your analysis
(2) accuracy and clarity of interpretation
(3) thoroughness of the design variations investigations and the documentation of the reasons for your assessments.

Summary report due: **October 26, 2023**