**Jonathan Samson**

**Brandon Chin**

**CMSC 409 – Project 3**

**1) What are the inputs and outputs for this problem.**

There is only one input and one output for this problem. The one input is the time in hours, and the one output is the hourly energy consumption in kW. However, the goal for this project is to find a polynomial regression that approximately matches the data. So, the inputs for the neuron will be slightly different than just the time of day. A polynomial function of degree 3 would look like *a+bx+cx2+dx3.* In this problem, we treat the time of day as *x*, and look for the different weights *a* through *d*, where *a* is the bias and b through c are weights 1 to 3. We have multiple inputs in this case, *x*, *x2*, and *x3*, where x is the time of day. We also create models for 1 degree, 2 degree, and 3 degree polynomial regression functions, and the inputs vary accordingly.

**2) What should be the activation function of your decision unit? Why did you choose it?**

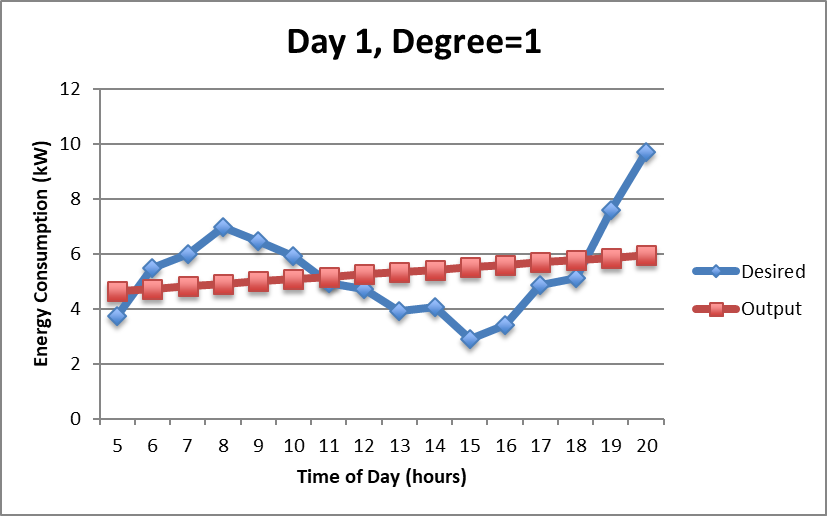
The activation function of the decision unit should be a linear activation function. This is because we are looking for continuous real values, rather than a decision function. In a normal decision problem, we would want values between 0 and 1 to classify the data, and would measure error as the distance from output to the correct answer. In this problem, the correct answer is not simply 0 or 1, but could be on a wide range of values measuring energy consumption. So we use a linear function which can go from negative infinity to positive infinity, in hopes our output will be close to the energy consumption.

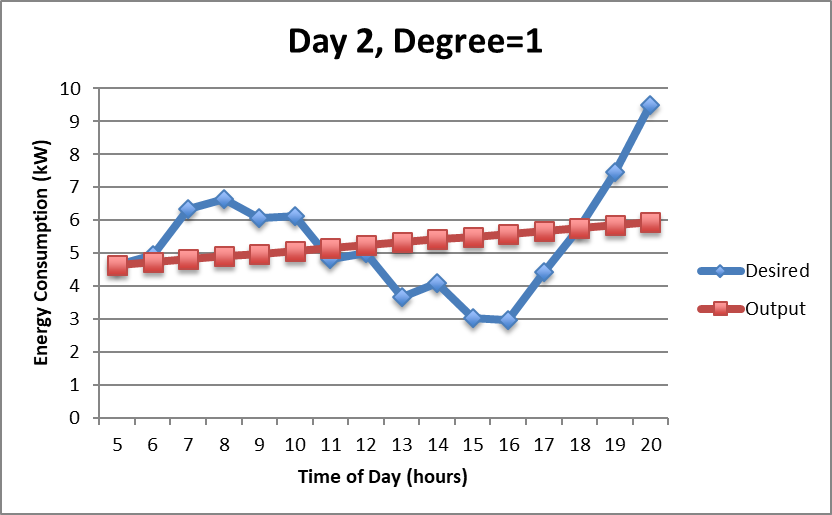
**3) Compare the training and testing errors obtained using the architectures on Figure 1**

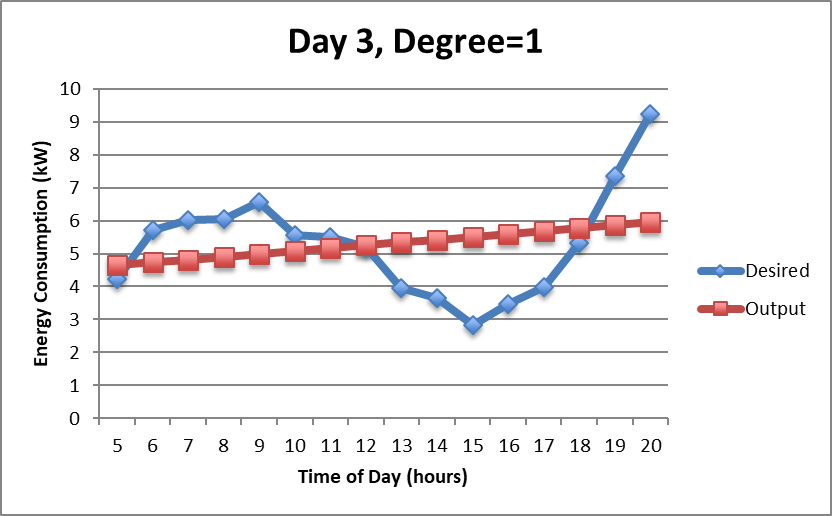
**a. Train the decision unit on the data from the first 3 days. Report training error for each of the three days. Present a graph (original data vs. trained model), similar to the Figure 2.**

We created graphs of the output and desired results for days 1 to 3, and models of degree 1 to 3. Total error is calculated as the square root of the sum of squared errors.

**Degree 1**

****



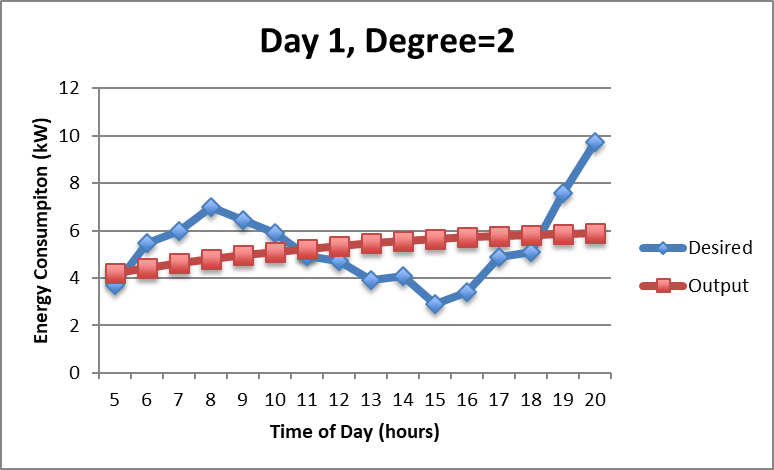
****

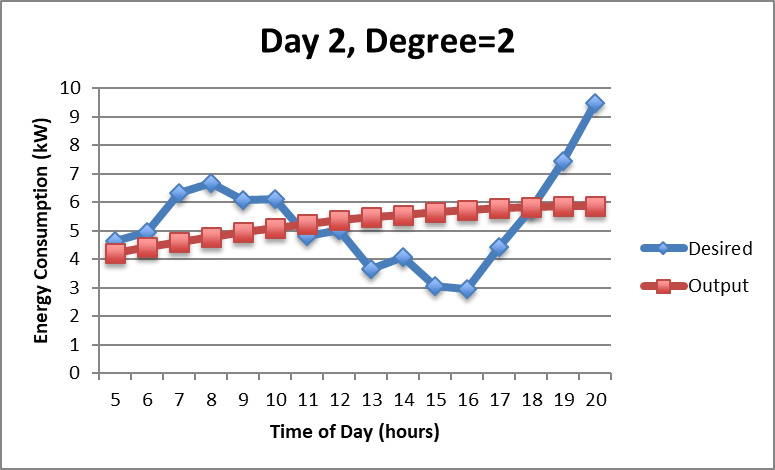
**Total Error for Day 1:** 6.649214787984489

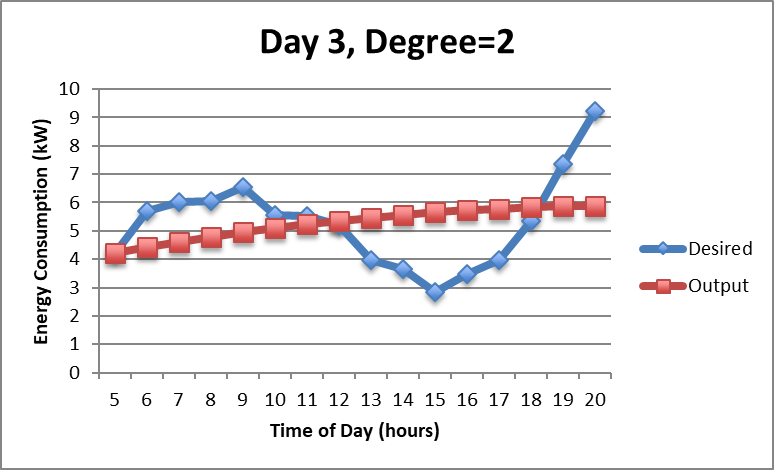
**Total Error for Day 2:** 6.5122696869325445

**Total Error for Day 3:** 6.318587571614721

**Degree 2**

****



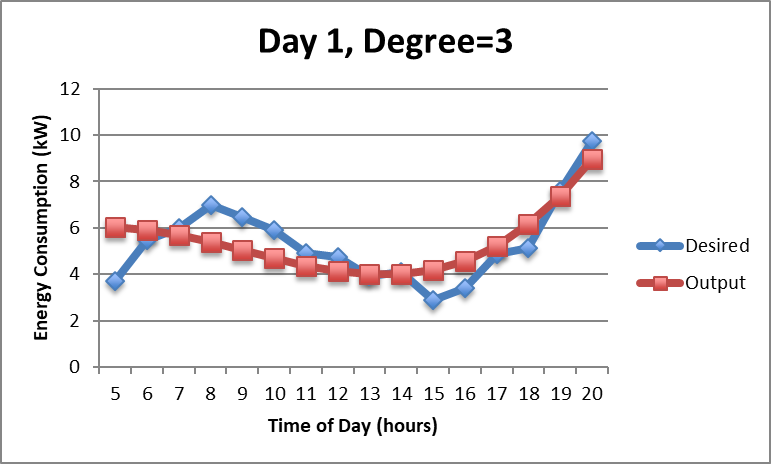


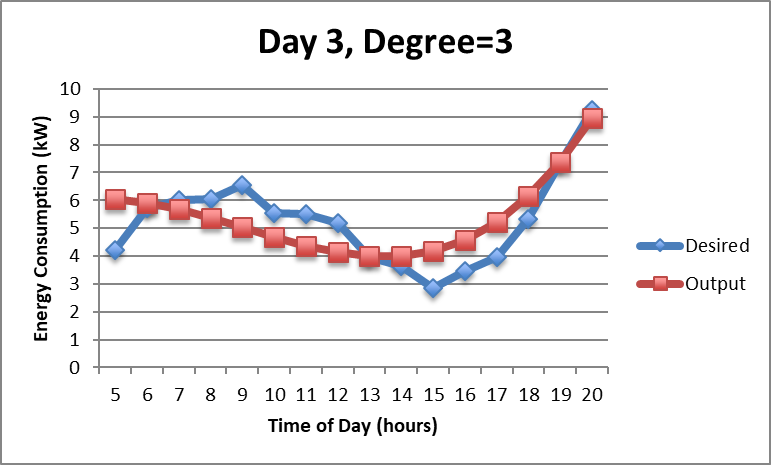
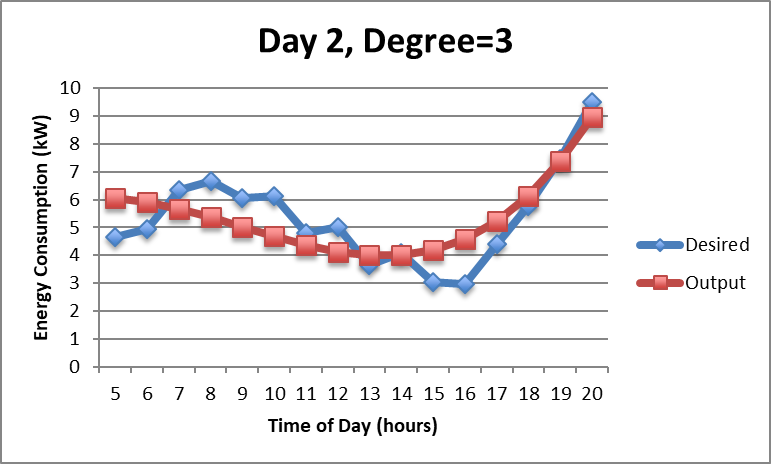
**Total Error for Day 1:** 7.3437976265969676

**Total Error for Day 2:** 7.283992162956873

**Total Error for Day 3:** 7.068096708100044

**Degree 3**

****

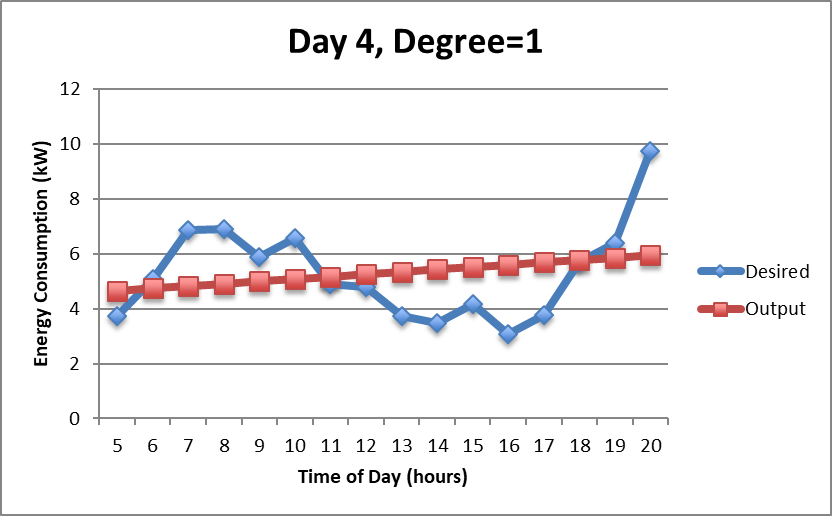
****

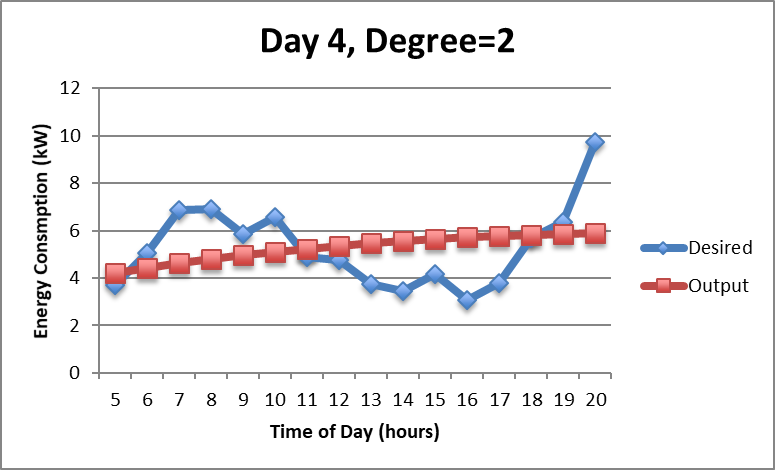
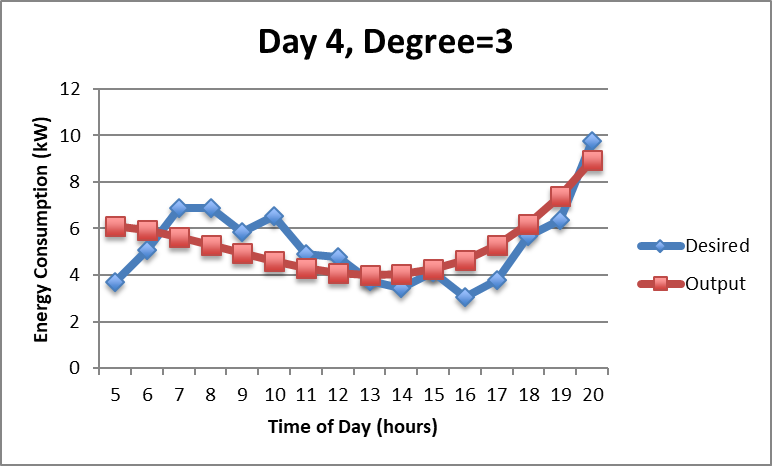
**Total Error for Day 1:** 4.1492729726042255

**Total Error for Day 2:** 3.758825869971608

**Total Error for Day 3:** 3.849389354715179

**b. Predict the energy consumption of the 4th date. Calculate the error of your prediction using the data for the 4th date. Report testing error. Present a graph (original data vs. trained model), similar to the Figure 2.**

**** 

**Total Error for Testing Data Degree 1:** 6.74502

**Total Error for Testing Data Degree 2:** 7.0456291

**Total Error for Testing Data Degree 3:** 4.84638328

**4) Report the number of iterations, the learning rate, data pre-processing steps you have chosen (such as normalization of input data). Clearly explain why you selected these values.**

The number of iterations we decided to have is 1,000,000, and our learning rate we decided to be .00000001. We needed to make sure the training constant is low, since the inputs can get very large, for example 203 is 8000. The weights will train too quickly and blow up if the training constant is not small enough. After testing both the training constant and the number of iterations at smaller and larger numbers, we found that too few iterations or too large of a training constant, and the output would be wildly inaccurate and exceedingly large. Using our knowledge from the previous assignments as well as significant guessing and checking, we decided that the numbers we chose resulted in the most accurate output. We tried with only 100 iterations as well as with a learning rate of 0.5, and we saw that the learning rate needed to be much lower, and the iterations needed to be much higher in order to achieve a more precise output.

There are several data preprocessing steps that can be taken including: data cleaning, data integration, data transformation, data reduction, and data discretization.

Data transformation is normalization and aggregation. We considered normalizing the data, and at first did try to train with normalized data. We would take the data pattern and divide each input by the length of the vector, to normalize the length to one. However when we did normalize the data, we obtained worse results. Normalizing in this way always made higher degree terms (*x3*) hover around .99, causing the weight training to have little effect on the output for that input. We decided not to normalize the data in the end.

Data cleaning is filling in missing values, smooth noisy data, identify or remove outliers, and resolve inconsistencies. Given the data sets that were given to us, we did not have to do any data cleaning with the data. Data integration is using multiple databases, data cubes, or files. We are using data integration with pulling training data from multiple files as well as using a separate test data file. We do not have multiple databases or data cubes. Data reduction is reducing the volume but producing the same or similar analytical results. Data discretization is part of data reduction, but it is replacing numerical attributes with nominal ones.

**5) Could the error be further reduced using a network of neurons (opposed to a single decision unit)? If so, discuss how and why these methods would reduce the error.**

Yes, we could easily build a 17 neuron system to approximate all 16 times of the day. Each of the first 16 neurons would take the time of day as input, and produce either a 0 or a 1, predicting whether a pattern comes from the respective day. The final neuron could take the 16 results as input, and guess the output. This system would be very accurate, as a separate weight would be trained for each time, and produce accurate outputs for each time.

The above idea shows one way we could use multiple neurons to reduce error. We can split the data into sections, and then use one neuron for each section determine whether the data lies in that section, then use a final neuron to make a final prediction. If we are doing a degree 3 polynomial, we can send 1, *x*, *x2*, and *x3* from the neuron who’s section contains the input, but zero from other neurons. The final neuron can have 4 weights (for each degree and a bias) for each section, to then predict the final result. For example, we could train one neuron to predict whether the input is between 5am and 12pm, and train another to predict whether the input is between 1pm and 8pm. A final neuron has 8 inputs. It receives 1, *x*, *x2*, and *x3* from the correct neuron, and zero from the other. For example, if the time is 7am, it receives 1, *x*, *x2*, and *x3* on inputs 1, 2, 3, and 4, and receives zero in inputs 5, 6, 7, and 8. Now only the first four weights factor into the final result. Using this system of neurons essential produces a piecewise approximation, which can be more accurate since we are using a different polynomial for each section of data. This system, however, would require more training. Firstly, we need more training to make the first layer of neurons accurate. Then we need more training since the final neuron has more weights that before.