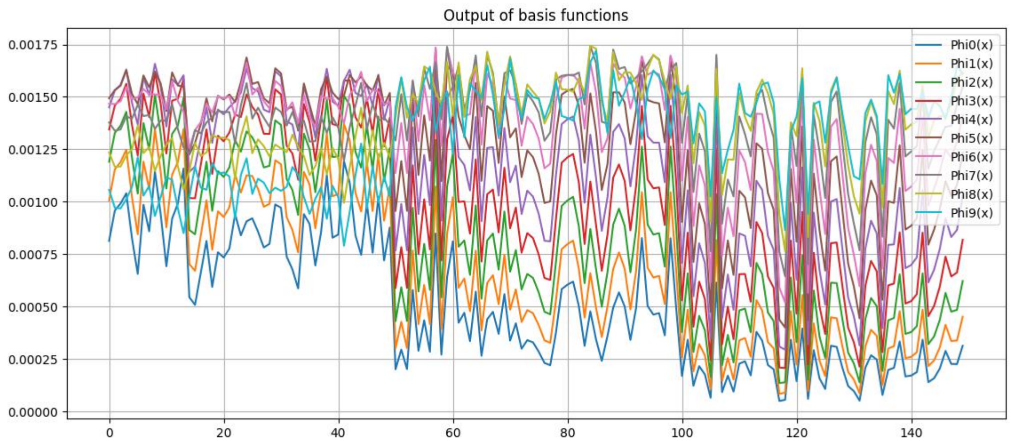
Sigurður Ágúst Jakobsson

Project: Assignment 7 – Linear Models for Regression

Section 1.2

Plot the output of each basis function, using the same parameters as above, as a function of the features. You should plot all the outputs onto the same plot. Include your plot as plot_1_2 in your PDF document.

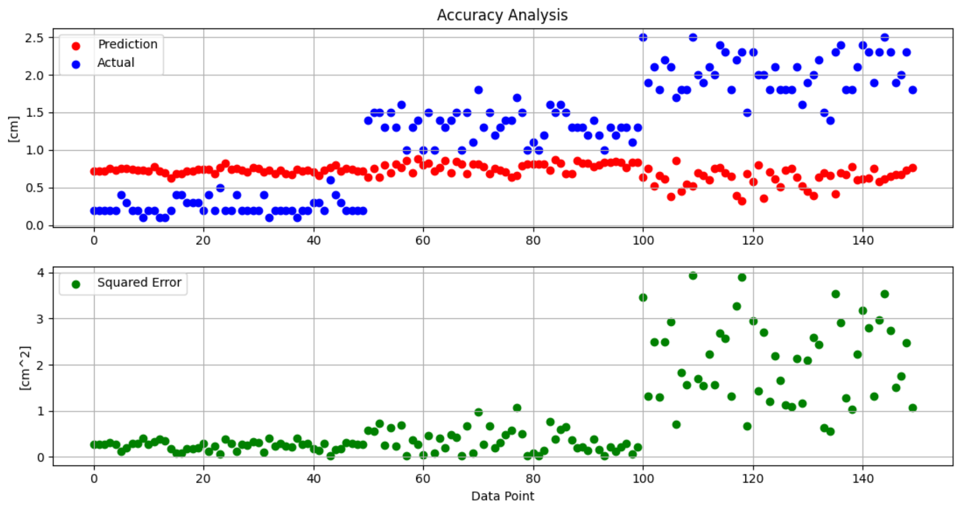
Here are all the outputs on the same plot.



Section 1.5

- How good are these predictions?
- Use plots to show the prediction accuracy, either by plotting the actual values vs predicted values or the mean-square-error.

On the figure below we can see the comparison of actual values vs. predicted values and the squared error for each point. We can see that the three classes are in sequence in the data since it is not shuffled. The predictions are not very good at all. They are usually very similar as they do not significantly change between classes. They do not provide good predictive value and can be compared with a constant guess regardless of the input features. The predictions are furthest off for the third class (label 2 in classification data). They even move in the wrong direction on average compared to the other classes (predict smaller than the other classes when they are really the largest).



Independent Section

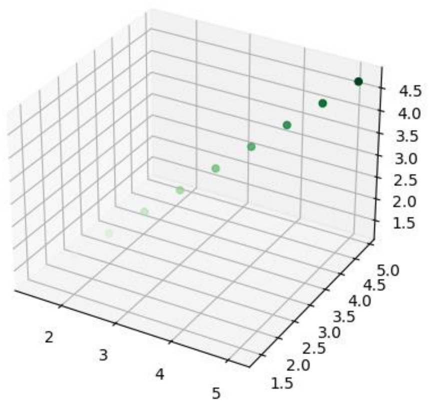
My goal for the independent section is to improve the functionality of the model by using a different number of basis functions and different mean vectors.

Firstly, I believe the goal of the given data was supposed to model a single vector through feature space to capture the change in size of the three features. I believe there was a small mistake however so that it did not capture the whole range of the variables.

This code takes the min and max of the first three rows of data not the columns:

```
M, sigma = 10, 10
mu = np.zeros((M, D))
for i in range(D):
    mmin = np.min(X[i, :])
    mmax = np.max(X[i, :])
    mu[:, i] = np.linspace(mmin, mmax, M)
```

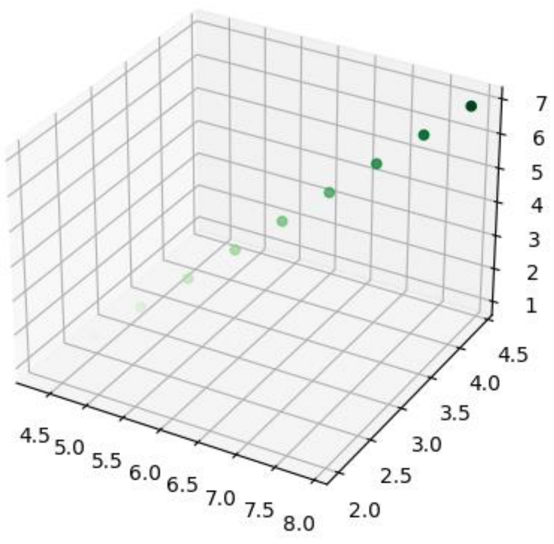
The rows and columns are mixed up if the goal is to take the min and max of the columns. This produces the following points in mu, which are all values from the first class:



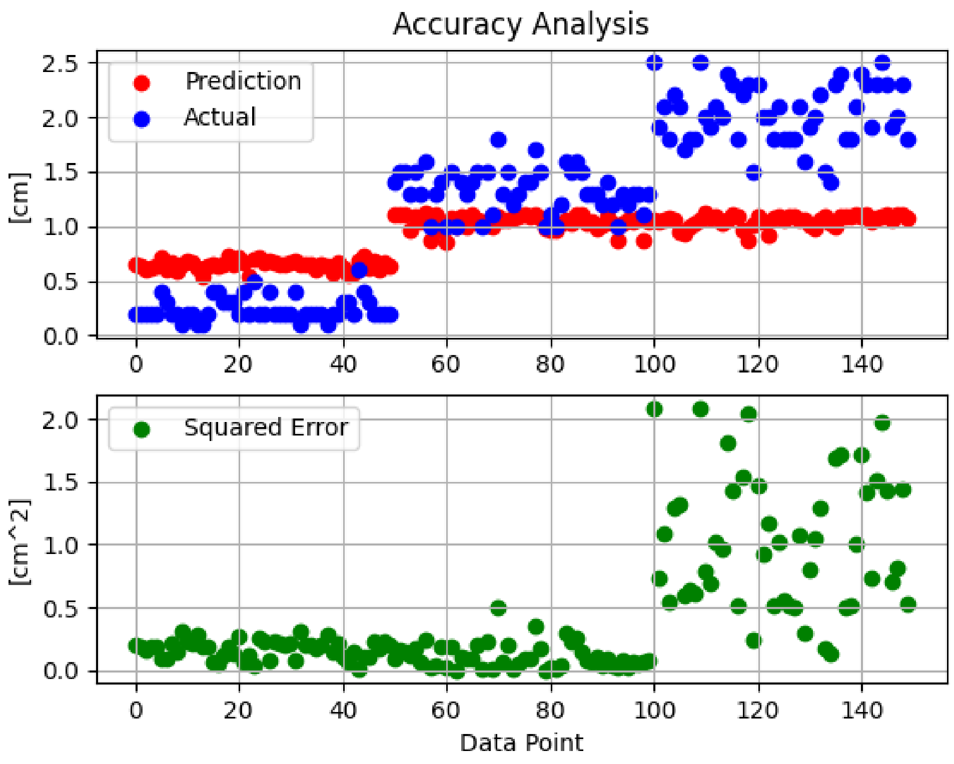
This immediately suggests a simple improvement to switch the rows and columns and try the training and prediction again. The function _prediction_accuracy_indep1() in the hand in does exactly this with the code:

```
    M, sigma = 10, 10
    mu = np.zeros((M, D))

    #Fixed to look and min and max of columns, not rows
    for i in range(D):
        mmin = np.min(X[:, i])
        mmax = np.max(X[:, i])
        mu[:, i] = np.linspace(mmin, mmax, M)
```

This produces a vector in feature space that can be seen below:



I believe this represents the variability in the features somewhat better and the predictions also look better as can be seen on the next plot.



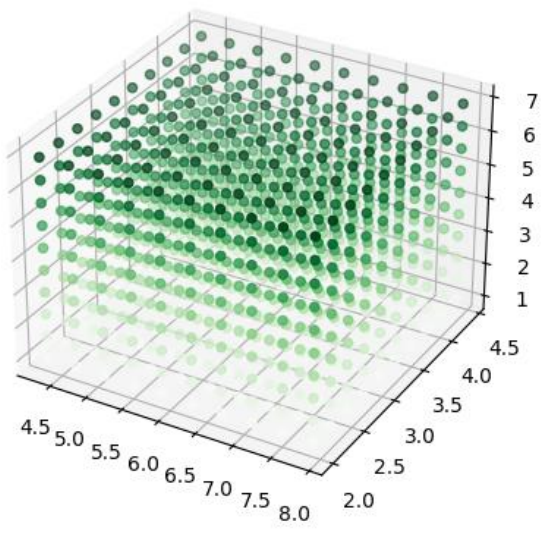
With this fix we see that the prediction can tell apart the first class from classes two and three. It can't tell apart the second and third class but the squared errors are much lower than before for the third class.

Now it would be interesting to see if we tried to capture the whole feature space. This is done in _prediction_accuracy_indep2() in the hand in.

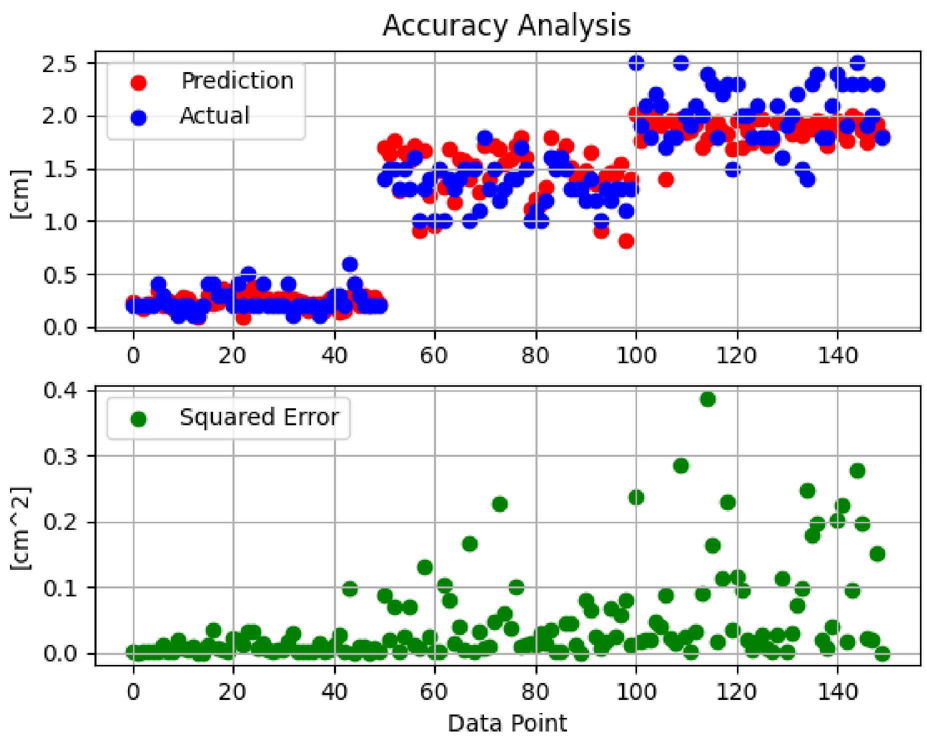
Here we build mu with the following code and use many more basis functions:

```
    points, sigma = 10, 10
    M = points**D
    feature_space = np.zeros((points, D))

    for i in range(D):
        mmin = np.min(X[:, i])
        mmax = np.max(X[:, i])
        feature_space[:, i] = np.linspace(mmin, mmax, points)

    mu = np.zeros((M, D))

    counter = 0

    for i in range(points):
        for j in range(points):
            for k in range(points):
                mu[counter] = np.array([feature_space[i, 0], feature_space[j, 1], feature_space[k, 2]])
                counter+=1
```

This results in the following setup within mu, where we cover much more of the feature space:



Now when we run the training and prediction, we can see the output below:



This is starting to look pretty good. Now the model can differentiate between all classes, the points line up relatively well, and the squared error for each class is much lower than before. We could probably get even better performance by dividing feature space into smaller intervals, but this is now starting to take 1-2 minutes to run since we are inverting a much larger matrix in the (lambda*I + phi_T * phi)^-1 term of w_bar = (lambda*I + phi_T * phi)^-1 * phi_T * t (MxM = 1000x1000). I think this is a conceptual proof of how to improve the model so I will leave it at this.