Question 2 – nonlinear regression

## Method 1 : knn

As a first non-linear method for solving the problem, I considered k-nearest neighbors algorithm because it is easier to implement and could see faster the results of a non-linear approach.

Unlike the assignments where we had to make classification using the knn and had labels on the last columns in the dataset, this time I had to adapt knn so it could deal with regression. I have chosen to sum the regression values for the k-nearest neighbours, to calculate the average of this sum and assign it for the test pattern.

In order to make a good generalization I have decided to use the cross-validation technique to find the best k that classifies the train patterns and then use it for the training dataset.

I have also chosen to implement the solution for this question in Matlab, and use mostly the code in the assignment because it was easier to adapt to the current question and I had better knowledge about how it works.

For the implementation I have gone through the following steps:

* After reading the data from files and then normalized it because, as I have noticed in the assignment, it ensures that each parameter counts in the same way for calculating the nearest neighbor.
* For the cross validation I have chosen to also split the dataset into 5 parts and I have taken values for k between 1 and 15 and incremented the value by 1. After each split I used my function knnAvg which takes a train set and a test set, for each element in test set calculates the regression value as described previously; at the end the root-mean-squared error between predictions of knn and values in last column of test set is computed and returned.
* After this, the k with lowest mean-squared error is considered and calculated the mean-squared error for train and test dataset by using the k that was found to be the best.

The results for running the code shown that the value of root-meant squared error is the best for and the value of average RMS during cross-validation is decreasing towards the largest k which means it could possibly be even lower for greater k values. However, for this value of k I have obtained mean-squared error is for the train set and for the test dataset.

As it can be seen the results of using knn to predict the regression value obtains error a lot larger than the linear approach. Algorithm I have chosen for knn to solve this problem could be improved: another approach for calculating the regression value could have been to calculate a weighted sum by the inverse of distance towards the k-nearest neighbors and also consider the distance towards the nearest neighbors and only takes those within a specific radius(perhaps even use different radius values for cross-validation). But since it is a very large difference from the RMS obtained by the linear regression approach, I have decided not to struggle much with knn and consider another algorithm for non-linear regression.

## Method 2: Random forests

As a second method for trying to obtain a better prediction for redshift method, I have chosen to use random forests. I have chosen this method mostly because it was not in the assignments, and I have read in many documentations that is one of the most efficient methods in Machine Learning and was excited to try something new, especially that the algorithm for random forest did not seem to be hard.

I have also implemented this in Matlab and I have implemented on my own the algorithm for building the random forest and calculate the final results but I have chosen to use the Matlab implementation *fitrtree* for Regression Threes for several reasons: I do not have good knowledge about how to create a tree data structure in Matlab and certainly it would have taken a lot of time to make it work properly, and also the algorithms are optimized in Matlab for building the trees and calculate prediction.

I have chosen to try several numbers of trees in the forest by using different number of regression trees and apply the cross validation technique to decide which number of trees is best for obtaining good results. From reading documentations it seems that by increasing the number of trees the accuracy of prediction is also increasing but since it was a new method for me I wanted to check to make sure I choose the number of trees that ensures the best generalization

I have implemented the algorithm by following the steps:

* I read data from files and decided not to make any normalization because, since one feature is never compared in magnitude to other features, the means and deviations do not matter.
* I have applied the 5-fold cross validation and I have chosen the numbers: 100,200 and 300 of trees. Perhaps I should have chosen more values but the computation time is very long and I wanted to get some results to check if the random forest may provide better predictions than the linear model
* At each step I created the random forest by implementing the algorithm shown in the course: the trees are created using the bootstrap sampling, and using the Matlab function *fitrtree* that creates objects of type *RegressionTree*. For each tree that was created I have provided the parameters of the patterns and target values, and also set off the *Prune* which is not used for random forests.
* After creating the trees predictions are calculated for the split from train data which is used as test. The method *predicts* returns prediction value for the specific *RegressionTree* and the input pattern from test. As I did for k-means I took the averaged sum of predictions from each tree and decided the final prediction. After this step the RMS between predictions and redshift column from the patterns is calculated and averaged between the sections of the cross validation.
* At the end I used the number of trees that provided the best mean-squared error and used it to calculate predictions for train and test data and see final results

The cross validation shown that there I not much difference in result between the numbers 100, 200 and 300 of trees but for 300 trees the random forest provided slightly better mean squared error. It is proven the theory that random forests provide better accuracy for larger number of trees but at some point the increase in accuracy is very small. I obtained a mean-squared error of 0.00072 for train data and 0.0012 for test data.

As a conclusion, the random forest performed better than the linear model and perhaps with more tuning of the model I could have obtained better results, because there are several parameters which can be changed and using cross validation it can be decided which provides the best results: the minimum size of terminal nodes and maximum number of terminal nodes. Unfortunately, because of the very long computation time and also long time I needed to understand the Matlab libraries for *RegressionTree* and how to make them work in my Random Forest, I did not have enough time to also check which values for these parameters provide the best results at cross validation but without these I have already obtained slightly better results and have proven that it is possible to obtain better performance with a non-linear regression model.