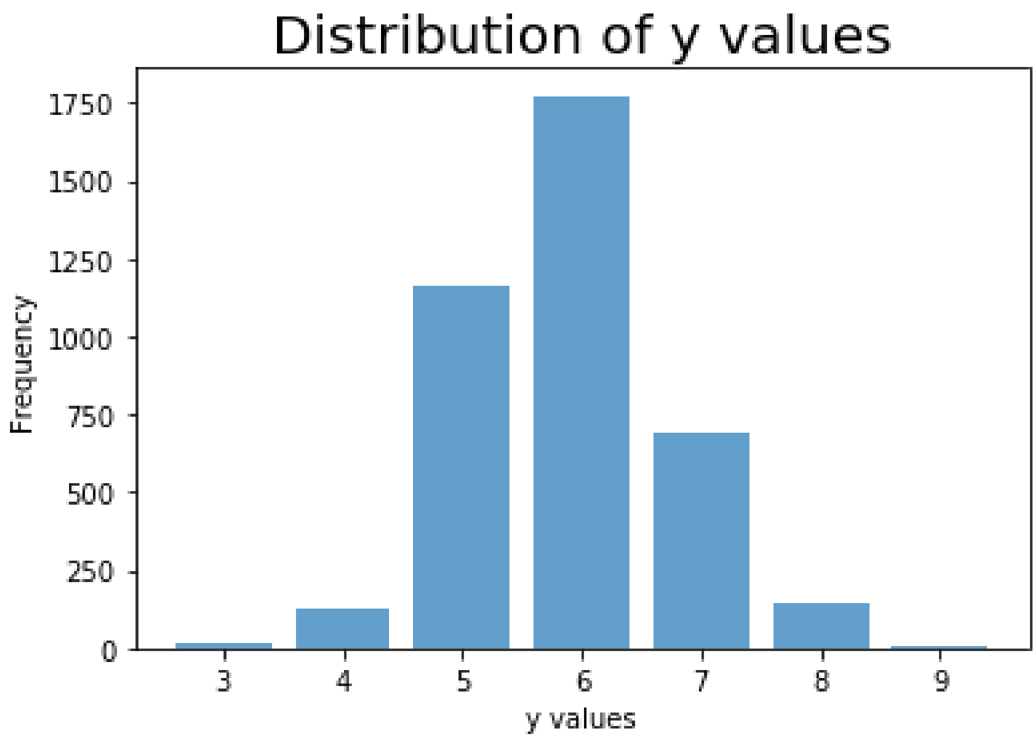
Practical1

November 17, 2017

In [1]:
```
%matplotlib inline
import _pickle as cp
import numpy as np
import numpy.linalg as linalg
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.model_selection import KFold

# Loading dataset
X, y = cp.load(open('winequality-white.pickle', 'rb'))

# Splitting the dataset into the training and test sets
N, D = X.shape
N_train = int(0.8 * N)

N_test = N - N_train

X_train = X[:N_train]
y_train = y[:N_train]

X_test = X[N_train:]
y_test = y[N_train:]
```

In [2]:
```
# Get the unique values of y and their corresponding frequencies
unique_y_train, counts = np.unique(y_train, return_counts = True)

# Plotting the distribution as a bar chart
plt.bar(unique_y_train, counts, align = 'center', alpha = 0.7)
plt.xlabel('y values')
plt.ylabel('Frequency')
plt.title('Distribution of y values', fontsize=20)

plt.show()
```



In [3]:
```
# Computing the simplest of predictors
y_mean_vector_train = np.repeat(np.mean(y_train), y_train.size)
y_mean_vector_test = np.repeat(np.mean(y_test), y_test.size)

# Computing the mean squared error (MSE) on the training set
squared_errors_vector_train = (y_train - y_mean_vector_train) ** 2
mse_train = np.mean(squared_errors_vector_train)

# Computing the mean squared error (MSE) on the test set
squared_errors_vector_test = (y_test - y_mean_vector_test) ** 2
mse_test = np.mean(squared_errors_vector_test)

print ('The MSE on training y-values with the mean is: ', mse_train)
print ('The MSE on test y-values with the mean is: ', mse_test)
```

```
The MSE on training y-values with the mean is:  0.77677723865
The MSE on test y-values with the mean is:  0.813839025406
```

In [4]:
```
# The standardization is not strictly necessary because there is no regularization term
# Hence, there is no strict requirement of weights to be present on a similar scale.

# Standardizing the training set
```

```
X_train_standardized = (X_train - np.mean(X_train, axis = 0)) / np.std(X_train, axis = 0

# Applying training set standardization transformation on test data
X_test_standardized = (X_test - np.mean(X_train, axis = 0)) / np.std(X_train, axis = 0)
```

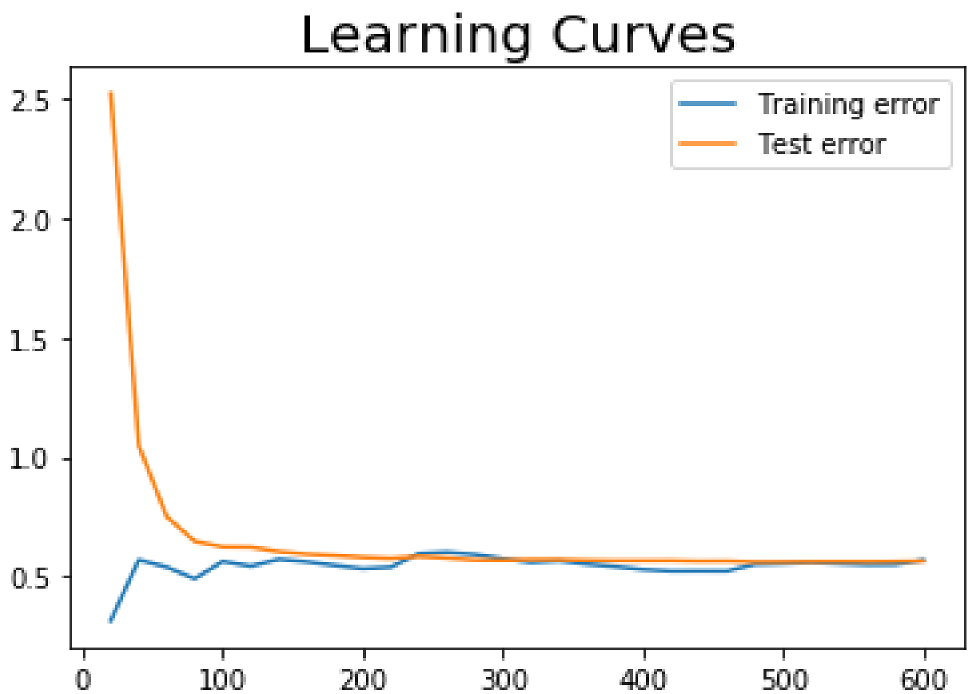
In [5]:
```
# In the following piece of code we try to fit a linear model to the data using the clos

# Step 1: Adding in a column of 1s to the training set
ones_column_train = np.ones((X_train_standardized.shape[0], 1))
ones_column_test = np.ones((X_test_standardized.shape[0], 1))
X_train_standardized = np.concatenate((ones_column_train, X_train_standardized), 1)
X_test_standardized = np.concatenate((ones_column_test, X_test_standardized), 1)
```

In [6]:
```
# Step 2: Closed form expression of the linear model inv((X'X))X'Y
W = (linalg.inv(X_train_standardized.T.dot(X_train_standardized))).dot(X_train_standardi

# Step 3: Computing MSE on training set
sq_errors_train = (X_train_standardized.dot(W) - y_train) ** 2
mse_train2 = np.mean(sq_errors_train)

# Step 4: Computing MSE on test set
sq_errors_test = (X_test_standardized.dot(W) - y_test) ** 2
mse_test2 = np.mean(sq_errors_test)

print ('The MSE obtained on training set with linear regression is: ', mse_train2)
print ('The MSE obtained on test set with linear regression is: ', mse_test2)
```

```
The MSE obtained on training set with linear regression is:  0.563999617394
The MSE obtained on test set with linear regression is:  0.560729204228
```

In [7]:
```
# Computing learning curves to detect over/underfitting

learning_curves = np.empty((30,2))
x_plot = np.empty((30,1))
z = 0
for i in range(20,620,20):
    # Create the step of training and test sets
    X_train_step = X_train_standardized[0:i,:]
    y_train_step = y_train[0:i]

    # Train linear model
    W_step = (linalg.inv(X_train_step.T.dot(X_train_step))).dot(X_train_step.T.dot(y_tra

    # Computing training set error
    sq_errors_train_step = (X_train_step.dot(W_step) - y_train_step) ** 2
    mse_train_step = np.mean(sq_errors_train_step)

    # Computing test set error
```

```
    sq_errors_test_step = (X_test_standardized.dot(W_step) - y_test) ** 2
    mse_test_step = np.mean(sq_errors_test_step)

    learning_curves[z,0] = mse_train_step
    learning_curves[z,1] = mse_test_step
    x_plot[z] = i
    z = z + 1

#print (learning_curves.shape)
plt.plot(x_plot, learning_curves[:,0], label = 'Training error')
plt.plot(x_plot, learning_curves[:,1], label = 'Test error')
plt.legend()
plt.title('Learning Curves', fontsize=20)
plt.show()

# How to check if the curve is underfitting or perfectly fitting?
```

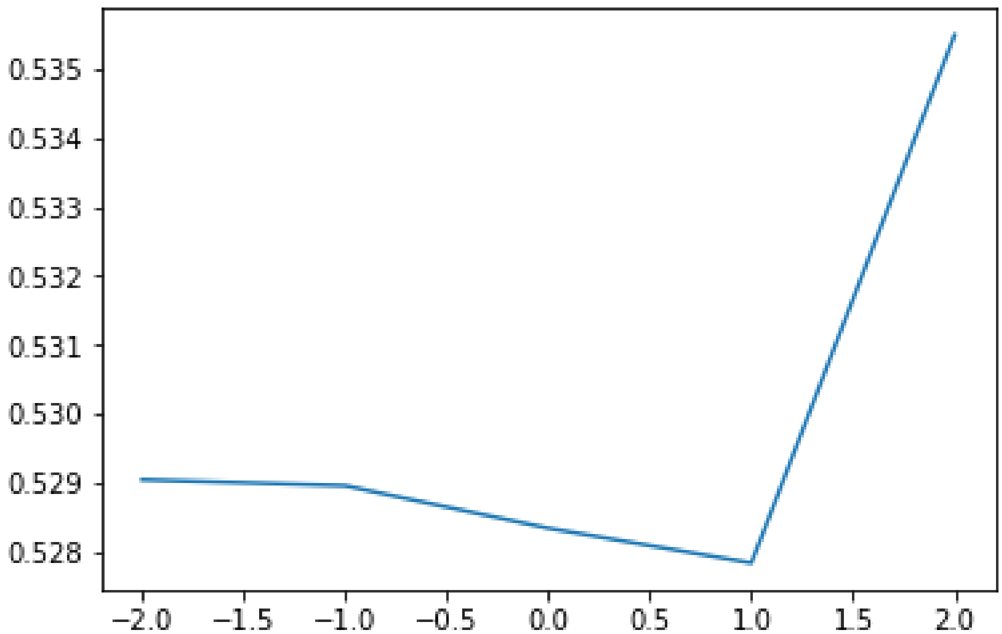


In [8]:
```
# Optional Task 1

# Step 1: Getting the training and validation sets
N, D = X_train.shape
N_train = int(0.8 * N)

X_train_expanded = X_train[:N_train]
```

```
y_train_expanded = y_train[:N_train]

X_validation_expanded = X_train[N_train:]
y_validation_expanded = y_train[N_train:]

# Step 2: Standardize the training and validation sets
scaler = StandardScaler()
scaler.fit(X_train_expanded)
X_train_expanded_standardized = scaler.transform(X_train_expanded)
X_validation_expanded_standardized = scaler.transform(X_validation_expanded)

poly = PolynomialFeatures(2)
X_train_expanded_standardized = poly.fit_transform(X_train_expanded_standardized)
X_validation_expanded_standardized = poly.fit_transform(X_validation_expanded_standardiz
```

In [9]:
```
# Getting optimal regularization parameter for ridge regression
x = []
y = []
l_best = 0
min_mse = 100

for i in range(-2,3,1):
    l = (10 ** i)
    ridge = Ridge(alpha = l)
    ridge.fit(X_train_expanded_standardized, y_train_expanded)
    y_validation_predicted = ridge.predict(X_validation_expanded_standardized)

    # Computing mean squared error
    sq_error = (y_validation_predicted - y_validation_expanded) ** 2
    mse = np.mean(sq_error)
    if (mse < min_mse):
        min_mse = mse
        l_best = l
    x = x + [i]
    y = y + [mse]

print (y)
plt.plot(x,y)
plt.show()
print ("Best value for lambda in ridge: ", l_best)
ridge_best_l = l_best
```

```
[0.52904090856545483, 0.52895512854365634, 0.52833904966043232, 0.52783399966993849, 0.535498211
```

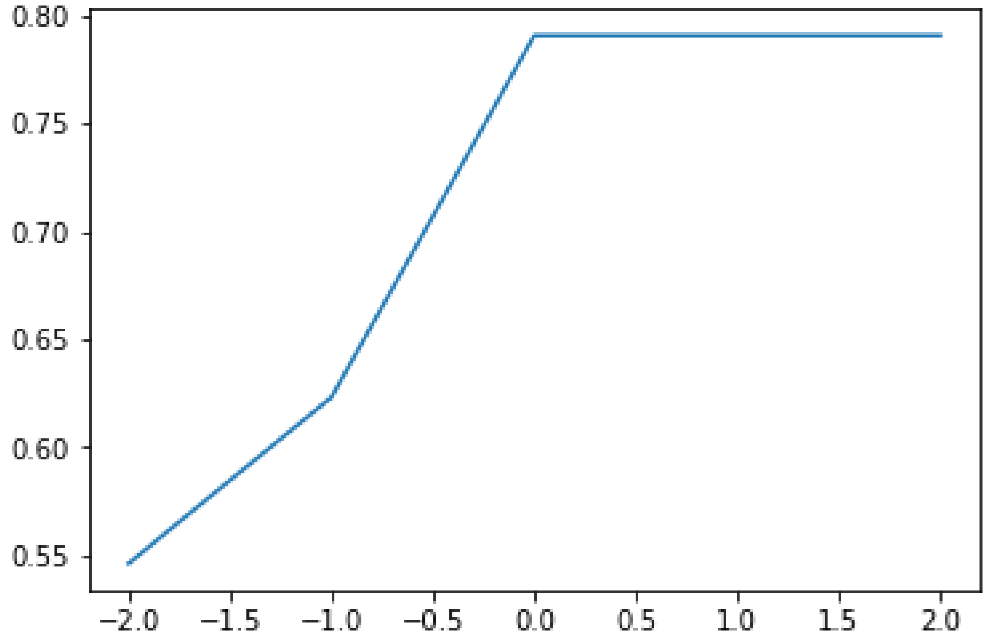



```
Best value for lambda in ridge:  10
```

```
In [10]: # Getting optimal regularization parameter for lasso regression
         x = []
         y = []
         l_best = 0
         min_mse = 100

         for i in range(-2,3,1):
             l = (10 ** i)
             lasso = Lasso(alpha = l)
             lasso.fit(X_train_expanded_standardized, y_train_expanded)
             y_validation_predicted = lasso.predict(X_validation_expanded_standardized)

             # Computing mean squared error
             sq_error = (y_validation_predicted - y_validation_expanded) ** 2
             mse = np.mean(sq_error)
             if (mse < min_mse):
                 min_mse = mse
                 l_best = l
             x = x + [i]
             y = y + [mse]

         print (y)
         plt.plot(x,y)
```

```
plt.show()
print ("Best value for lambda in lasso: ", l_best)
lasso_best_l = l_best
```

```
[0.54629378923215255, 0.62320714726892057, 0.79063388334113016, 0.79063388334113016, 0.79063388
```




```
Best value for lambda in lasso:  0.01
```

In [11]:
```
# Standardize the training set
sc = StandardScaler()
scaler.fit(X_train)
X_train_2 = scaler.transform(X_train)
X_test_2 = scaler.transform(X_test)

X_train_2_expanded = poly.fit_transform(X_train_2)
X_test_2_expanded = poly.fit_transform(X_test_2)
```

In [12]:
```
# Ridge with optimal lambda on training and test set
ridge_optimal = Ridge(alpha = ridge_best_l)
ridge_optimal.fit(X_train_2_expanded, y_train)
y_test_predicted = ridge_optimal.predict(X_test_2_expanded)

ridge_mse = np.mean((y_test_predicted - y_test) ** 2)
print ("Ridge optimal MSE: ", ridge_mse)
```

```
Ridge optimal MSE:  0.511667745658
```

```
In [13]: # Lasso with optimal lambda on training and test set
         lasso_optimal = Lasso(alpha = lasso_best_l)
         lasso_optimal.fit(X_train_2_expanded, y_train)
         y_test_predicted = lasso_optimal.predict(X_test_2_expanded)

         lasso_mse = np.mean((y_test_predicted - y_test) ** 2)
         print ("Lasso optimal MSE: ", lasso_mse)
```

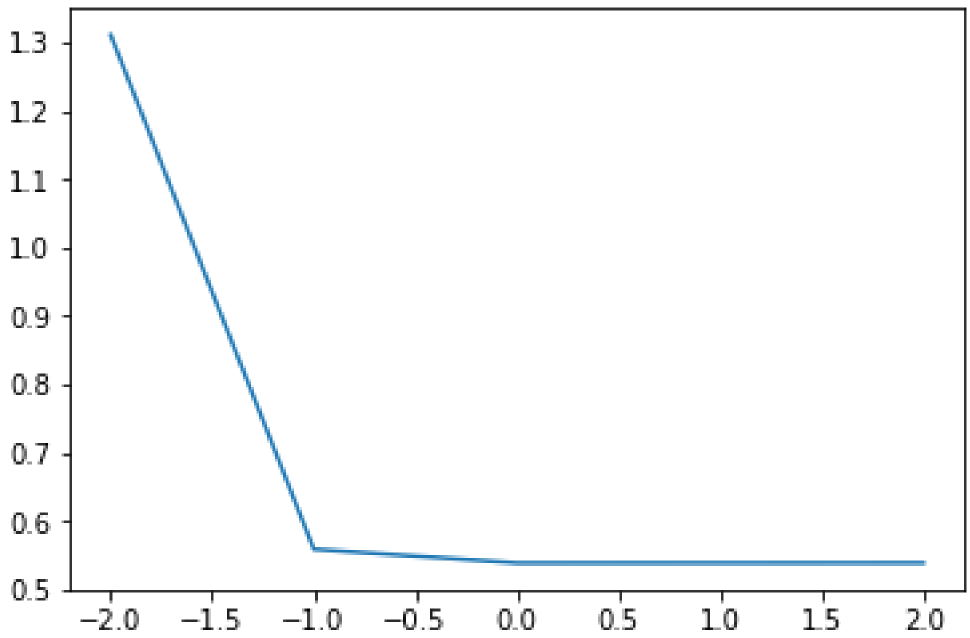
```
Lasso optimal MSE:  0.518552303917
```

```
In [14]: # Optional Task 2: Trying Ridge regression with basis expansion to the nth degree (n ta
         # and k-fold cross-validation (k also taken as a parameter)

         def ridge_with_cross_validation (X_train, y_train, X_test, y_test, l = 1, n = 2, k = 5)
             ''' The method performs ridge regression on the given dataset. The choice of hyperp
             varies from 10^-l to 10^l, there is nth degree polynomial expansion of the input an
             k-fold cross-validation.'''

             # 1. Generate cross validation folds
             k_fold = KFold(n_splits = k)
             p = []
             q = []

             best_l = 0
             min_err = 100

             # 2. Iterating for different regularization parameters
             for i in range(-l,l+1,1):
                 reg_param = 10^i;
                 err = 0

                 # 3. Iterating over each split
                 for train_index, test_index in k_fold.split(X_train):
                     x_train_val, x_test_val = X_train[train_index], X_train[test_index]
                     y_train_val, y_test_val = y_train[train_index], y_train[test_index]

                     # Standardize
                     sc = StandardScaler()
                     sc.fit(x_train_val)
                     x_train_val_std = sc.transform(x_train_val)
                     x_test_val_std = sc.transform(x_test_val)

                     # Basis expansion
                     poly = PolynomialFeatures(n)
```

```
            x_train_val_std = poly.fit_transform(x_train_val_std)
            x_test_val_std = poly.fit_transform(x_test_val_std)

            ridge = Ridge(alpha = reg_param)
            ridge.fit(x_train_val_std, y_train_val)
            y_test_val_predicted = ridge.predict(x_test_val_std)
            mse = np.mean((y_test_val_predicted - y_test_val) ** 2)
            err = err + mse

        err = err/k
        p = p + [i]
        q = q + [err]

        if (err < min_err):
            best_l = reg_param
            min_err = err

    print(q)
    plt.plot(p,q)
    plt.show()
    print("Best regularisation parameter: ", 10^best_l)

    # 4. Try fitting your linear model using best_l
    sc = StandardScaler()
    sc.fit(X_train)
    X_train_std = sc.transform(X_train)
    X_test_std = sc.transform(X_test)

    poly = PolynomialFeatures(n)
    X_train_std = poly.fit_transform(X_train_std)
    X_test_std = poly.fit_transform(X_test_std)

    ridge = Ridge(alpha = best_l)
    ridge.fit(X_train_std, y_train)
    y_test_predicted = ridge.predict(X_test_std)
    mse = np.mean((y_test_predicted - y_test) ** 2)
    print ("Overall MSE: ", mse)
    return mse
```

```
In [15]: ridge_with_cross_validation(X_train, y_train, X_test, y_test, 2, 2, 5)
```

```
[1.310735822308001, 0.55851509908079788, 0.53851148700042428, 0.53856789280409689, 0.53863238171
```

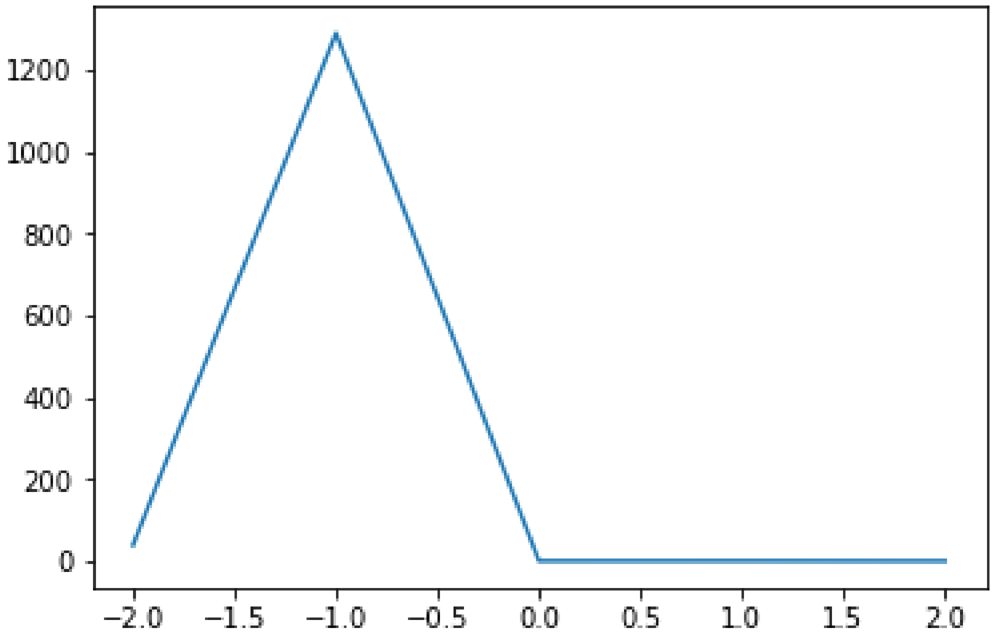



```
Best regularisation parameter:  0
Overall MSE:  0.511667745658
```

Out[15]: 0.5116677456584301

```
In [16]: ridge_with_cross_validation(X_train, y_train, X_test, y_test, 2, 3, 5)
```

```
[39.930657994293497, 1289.559035435017, 1.6017041972298025, 1.7041076465353799, 1.38346943385210
```

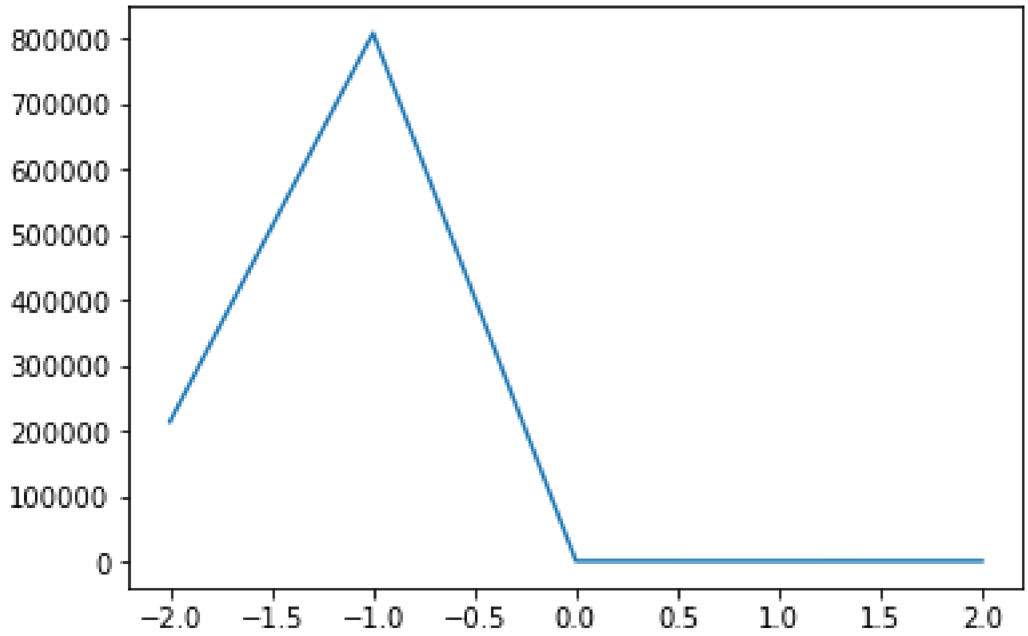



```
Best regularisation parameter:  2
Overall MSE:  0.652691045475
```

Out[16]: 0.65269104547497725

In [17]:
```
ridge_with_cross_validation(X_train, y_train, X_test, y_test, 2, 4, 5)
```

```
[214605.6098589777, 808947.22215526458, 1486.9644862245596, 1462.3032058234471, 1540.75363736474
```

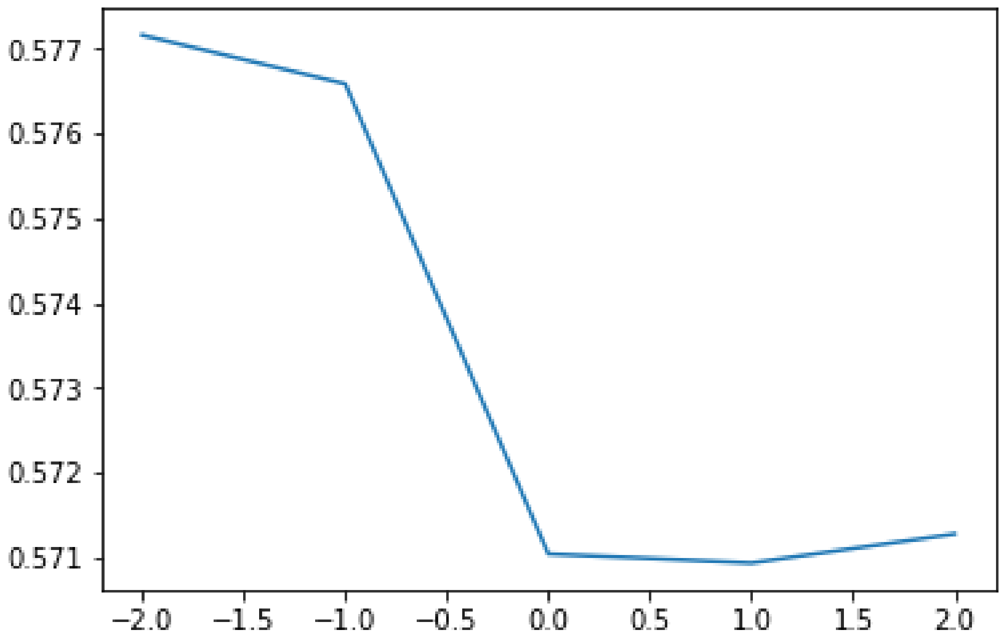



```
Best regularisation parameter:  10
Overall MSE:  3.93942721758
```

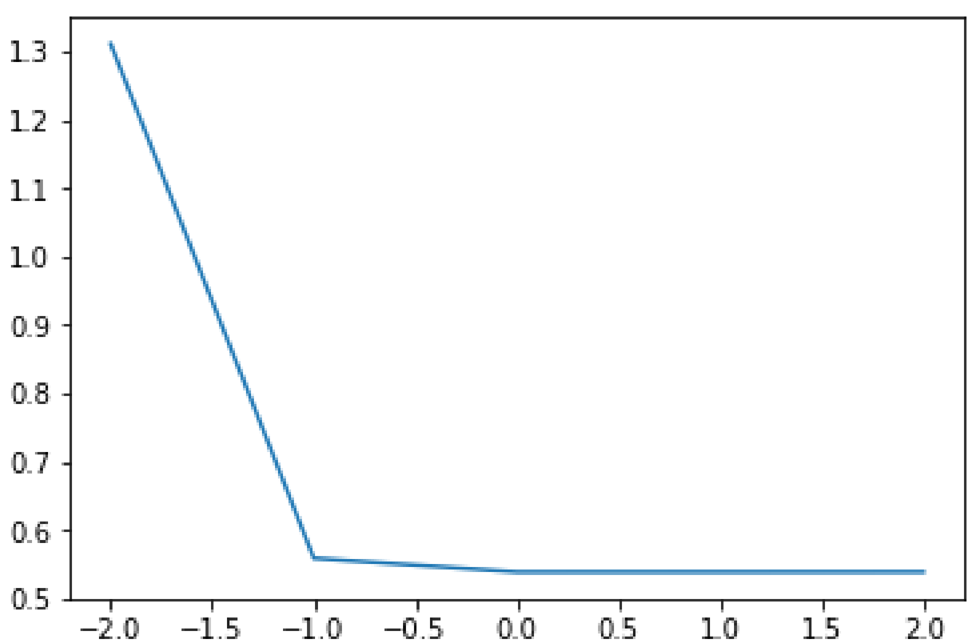
```
Out[17]: 3.9394272175844556
```

```
In [18]: x = []
         y = []
         for i in range(1,5,1):
             print("Basis expansion degree: ", i)
             x = x + [i]
             y = y + [ridge_with_cross_validation(X_train, y_train, X_test, y_test, 2, i, 5)]

         print ("-----------------------------------------------------------------------")
         plt.plot(x,y)
         plt.xlabel("Basis expansion degree")
         plt.ylabel("Overall RMSE error")
         plt.show()
```

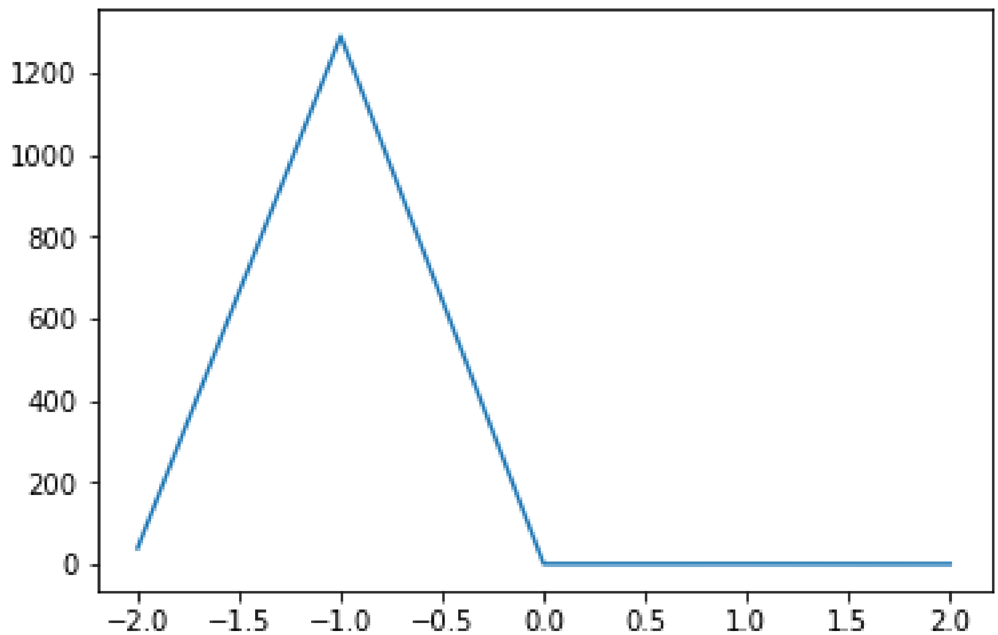
```
Basis expansion degree:  1
[0.57715926256622707, 0.5765838171277472, 0.57104041992119214, 0.57093417256387247, 0.5712762832
```



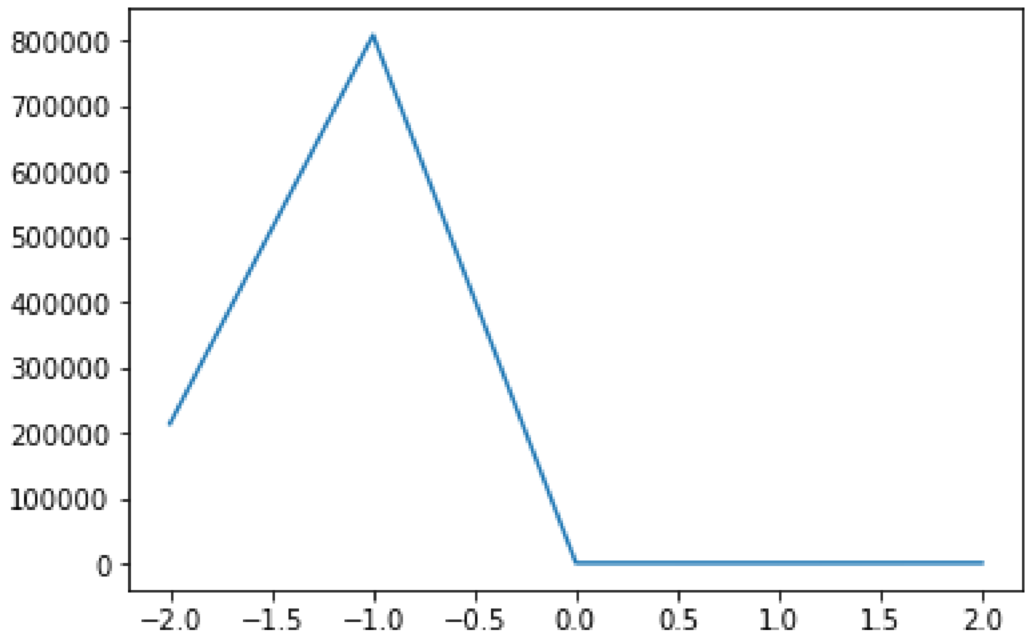

```
Best regularisation parameter:  1
Overall MSE:  0.560911558226
Basis expansion degree:  2
[1.310735822308001, 0.55851509908079788, 0.53851148700042428, 0.53856789280409689, 0.53863238171
```



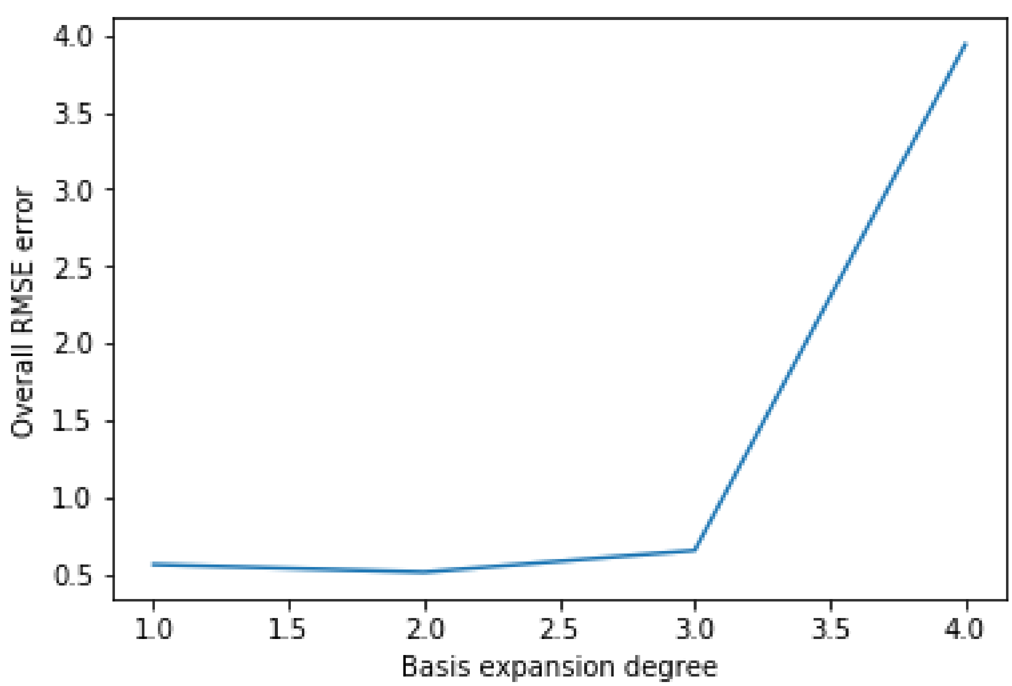
```
Best regularisation parameter:  0
Overall MSE:  0.511667745658
Basis expansion degree:  3
[39.930657994293497, 1289.559035435017, 1.6017041972298025, 1.7041076465353799, 1.38346943385210
```




```
Best regularisation parameter:  2
Overall MSE:  0.652691045475
Basis expansion degree:  4
[214605.6098589777, 808947.22215526458, 1486.9644862245596, 1462.3032058234471, 1540.75363736474
```




```
Best regularisation parameter:  10
Overall MSE:  3.93942721758
--------------------------------------------------------------------
```



```
In [ ]:
```