Gabriel Van Dreel
800886655
September 23, 2021
ECGR 4105-C01

Homework 1

The source code developed for homework 1 was uploaded to the Github repository at the following link:

https://github.com/Gabrielvd616/ECGR4105/tree/main/Homework1

Problem 1

Part a

- The gradient descent training and cost evaluation for the regressive model predicting housing price was developed with consideration of the area, bedrooms, bathrooms, stories, and parking input variables. The best input variables for the linear regression model were observed to be the bedrooms, bathrooms, stories, and parking variables where the area variable was not used because it resulted in very high values of the mean squared error loss function that did not converge. This likely resulted from the area variable having a much larger range of values relative to those of the other input variables. The model parameter corresponding to the bathrooms input variable was observed to be the best parameter for minimizing the training and validation (test) loss or cost.

  Let x_1 be the area, x_2 be the bedrooms, x_3 be the bathrooms, x_4 be the stories, and x_{10} be the parking input variables.

  The linear model for predicting the housing price was found and recorded in section [1] with respect to x_2, x_3, x_4, and x_{10} as shown below.

  $$y = 870489.6391051807 + 305563.8189222792x_2 + 1203712.2817691953x_3$$

  $$+561587.6336152942x_4 + 602856.1894135593x_{10}$$

  The training and test cost histories for the above model were graphed in section [2].

Part b

- The gradient descent training and cost evaluation for the regressive model predicting housing price was developed with consideration of the area, bedrooms, bathrooms, stories, mainroad, guestroom, basement, hotwaterheating, airconditioning, parking, and prefarea input variables. The best input variables for the linear regression model were observed to be the bedrooms, bathrooms, stories, mainroad, guestroom, basement, hotwaterheating, airconditioning, parking, and prefarea variables. The area variable was also not used because it caused the loss function to not converge. The model parameter corresponding to the bathrooms input variable was also observed to be the best parameter for minimizing the training and test cost.

Let x_5 be the mainroad, x_6 be the guestroom, x_7 be the basement, x_8 be the hotwaterheating, x_9 be the airconditioning, and x_{11} be the prefarea input variables.

The linear model for predicting the housing price was found and recorded in section [3] with respect to x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, and x_{11} as shown below.

$$y = 437633.94479189994 + 181036.77636442526x_2 + 988613.7604707936x_3$$

$$+410311.3402010583x_4 + 756461.4261511914x_5 + 538943.5229735046x_6$$

$$+265720.24743397126x_7 + 1180856.4100370929x_8 + 1088780.0307373086x_9$$

$$+413179.7633270081x_{10} + 978459.2987293216x_{11}$$

The training and test cost histories for the above model were graphed in section [4].

Problem 2

Part a

- The gradient descent training and cost evaluation for the regressive model predicting housing price was repeated with normalization (MIN MAX scaling) of the x_1, x_2, x_3, x_4, and x_{10} input variables. The linear model for predicting the housing price was found and recorded in section [5] with respect to x_1, x_2, x_3, x_4, and x_{10} as shown below.

  $$y = 0.04160884 + 0.36839377x_1 + 0.09341937x_2 + 0.20038078x_3$$

  $$+0.13404017x_4 + 0.09696011x_{10}$$

  The training and test cost histories for the above model were graphed in section [6].

  Similarly, the gradient descent training and cost evaluation for the regressive model predicting housing price was repeated with standardization of the x_1, x_2, x_3, x_4, and x_{10} input variables. The linear model for predicting the housing price was found and recorded in section [7] with respect to x_1, x_2, x_3, x_4, and x_{10} as shown below.

  $$y = 1.94574598e-16 + 3.83653304e-01x_1 + 1.04343457e-01x_2 + 2.98541735e-01x_3$$

  $$+2.34542828e-01x_4 + 1.49757135e-01x_{10}$$

  The training and test cost histories for the above model were graphed in section [8].

  The training accuracy of the normalization scaling approach was observed to be higher than that of the standardization scaling approach with both lower training and test cost. The normalization scaling may have resulted in lower cost because the training data correlated with the housing price may not have followed a normal distribution. The baseline training accuracy of the regression model that did not utilize scaling was observed to be significantly lower than those of both models that utilized different scaling methods in terms of training and test cost. This likely corresponded to much faster convergence of the cost history for the regression models that utilized scaling methods. The use of scaling also allowed for the use of the feature x_1 since a regression could not be performed without scaling the input variable, which was also observed to improve the accuracy of the scaled regression models by accounting for an additional independent variable.

Part b

- The gradient descent training and cost evaluation for the regressive model predicting housing price was repeated with normalization of the x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, and x_{11} input variables. The linear model for predicting the housing price was found and recorded in section [9] with respect to x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, and x_{11} as shown below.

  $y = -0.00030068 + 0.24463399x_1 + 0.0558249x_2 + 0.166414x_3$

  $+0.10983887x_4 + 0.04724366x_5 + 0.03795568x_6 + 0.02992177x_7$

  $+0.10666471x_8 + 0.08271132x_9 + 0.07520951x_{10} + 0.06660711x_{11}$

  The training and test cost histories for the above model were graphed in section [10].

  Similarly, the gradient descent training and cost evaluation for the regressive model predicting housing price was repeated with standardization of the x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, and x_{11} input variables. The linear model for predicting the housing price was found and recorded in section [11] with respect to x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, and x_{11} as shown below.

  $y = 1.91576122e - 16 + 2.69430270e - 01x_1 + 5.88474875e - 02x_2 + 2.48153203e - 01x_3$

  $+1.99265962e - 01x_4 + 9.20205597e - 02x_5 + 8.25960172e - 02x_6 + 9.41683856e - 02x_7$

  $+1.40206929e - 01x_8 + 2.32616682e - 01x_9 + 1.16157210e - 01x_{10}$

  $+1.62960286e - 01x_{11}$

  The training and test cost histories for the above model were graphed in section [12].

  The training accuracy of the normalization scaling approach was also observed to be higher than that of the standardization scaling approach with both lower training and test cost. The normalization scaling may have resulted in lower cost because the training data correlated with the housing price may not have followed a normal distribution. The training and test cost of the regression models using normalization and standardization was observed to decrease from those of the models shown in section [6] and section [8], which likely occurred because the models shown in section [10] and section [12] accounted for more independent variables. The baseline training accuracy of the regression model that did not utilize scaling was also observed to be significantly lower than those of both models that utilized different scaling methods in terms of training and test cost. This likely corresponded to much faster convergence of the cost history for the regression models that utilized scaling methods. The use of scaling also allowed for the use of the feature x_1 since a regression could not be performed without scaling the input variable, which was observed to improve the accuracy of the scaled regression models similarly to the scaled models with less independent variables.

Problem 3

Part a

- The gradient descent training and cost evaluation for the regressive model predicting housing price was developed with input normalization and regularization via parameter penalization. Normalization scaling was performed with the x_1, x_2, x_3, x_4, and x_{10} input variables because it was observed to have the least loss in problem 2. To implement the training regularization, a new function called gradient_descent_reg() was written that passed an additional argument of the regularization parameter reg_param. While the training cost history was newly calculated using the gradient_descent_reg() function, the test cost history was calculated using the original gradient_descent() function to assess loss without the influence of reg*param. The linear model for predicting the housing price was found and recorded in section [13] with respect to $x*{1},x*{2}*, x*{3}*,x*{4}*, $andx${10}$ as shown below.

  $$y = 0.04168137 + 0.36808029x_1 + 0.0934267x_2 + 0.20031399x_3$$

  $$+0.13403137x_4 + 0.09699192x_{10}$$

  The training and test cost histories for the above model were graphed in section [14].

  The training cost history for the above regression model using normalization scaling and regularization was observed to be greater than that of the regression model using normalization scaling without regularization with the same number of independent variables as expected with the addition of reg_param to the training and cost calculations. However, the test cost history for the above regression model was observed to be approximately equal to that of the regression model using normalization scaling without regularization, which suggests that the data used for the model likely correlated relatively closely with the housing price and therefore provided meaningful information to the model. With the learning rate $\alpha = 0.07$ and reg_param values ranging from 0.001 to 0.01, the test cost history was observed to approach approximately 0.00544 after $n = 1500$ iterations.

Part b

- The gradient descent training and cost evaluation for the regressive model predicting housing price was also developed with input normalization and regularization via parameter penalization using the x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, and x_{11} input variables. Using the same scaling and regularization methods as those used in section [13] and section [14], the linear model for predicting the housing price was found and recorded in section [15] with respect to x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, and x_{11} as shown below.

  $$y = -0.00025713 + 0.24441419x_1 + 0.05583525x_2 + 0.166358x_3$$

  $$+0.10982188x_4 + 0.04725399x_5 + 0.03796283x_6 + 0.02991754x_7$$

$$+0.1066072x_8 + 0.08271691x_9 + 0.07522892x_{10} + 0.06661762x_{11}$$

The training and test cost histories for the above model were graphed in section [16].

Similarly, the training cost history for the above regression model using normalization scaling and regularization was observed to be greater than that of the regression model using normalization scaling without regularization with the same number of independent variables as expected with the addition of reg_param to the training and cost calculations. However, the test cost history for the above regression model was observed to be approximately equal to that of the regression model using normalization scaling without regularization, which also suggests that the data used for the model likely correlated relatively closely with the housing price and therefore provided meaningful information to the model. With the learning rate $\alpha = 0.07$ and reg_param values ranging from 0.001 to 0.01, the test cost history was observed to approach approximately 0.00400 after $n = 1500$ iterations.

In [1]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler

'''
Computes gradient descent model parameters and cost for linear regression

Input parameters:
x : m x n np array of training samples where m is the number of training samples and n
    is the number of model parameters
y : m x 1 np array of training labels
theta : 1 x n np array of model parameters
alpha : scalar value for learning rate
iterations : scalar value for the number of iterations

Output parameters:
theta : 1 x n np array of final model parameters
cost_history : 1 x iterations array of cost values for each iteration
'''


def gradient_descent(x, y, theta, alpha, iterations):
    cost_history = np.zeros(iterations)

    for i in range(iterations):
        # Computes ML model prediction using column vector theta and x values using
        # matrix vector product
        predictions = x.dot(theta)
        errors = np.subtract(predictions, y)
        sum_delta = (alpha / m) * x.transpose().dot(errors)
        theta = theta - sum_delta

        # Computes value of cost function J for each iteration
        sqr_error = np.square(errors)
        cost_history[i] = 1 / (2 * m) * np.sum(sqr_error)

    return theta, cost_history
```

```python
'''
Computes gradient descent model parameters and cost for linear regression using
regularization method of parameter penalization

Input parameters:
x : m x n np array of training samples where m is the number of training samples and n
    is the number of model parameters
y : m x 1 np array of training labels
theta : 1 x n np array of model parameters
alpha : scalar value for learning rate
iterations : scalar value for the number of iterations
reg_param : scalar value for the regularization parameter

Output parameters:
theta : 1 x n np array of final model parameters
cost_history : 1 x iterations array of cost values for each iteration
'''


def gradient_descent_reg(x, y, theta, alpha, iterations, reg_param):
    cost_history = np.zeros(iterations)

    for i in range(iterations):
        # Computes ML model prediction using column vector theta and x values using
        # matrix vector product
        predictions = x.dot(theta)
        errors = np.subtract(predictions, y)
        sum_delta = (alpha / m) * (x.transpose().dot(errors) + reg_param * theta)
        theta = theta - sum_delta

        # Computes value of cost function J for each iteration
        sqr_error = np.square(errors)
        cost_history[i] = 1 / (2 * m) * (np.sum(sqr_error) + reg_param *
                                         np.sum(np.square(theta)))

    return theta, cost_history


# Defines the map function to map strings to numbers in table
def binary_map(x):
    return x.map({'yes': 1, 'no': 0})


# Problem 1, Part a
# Initializes the number of iterations and the learning rate alpha
iterations = 1500
alpha = 0.07

# Reads labelled training data
df = pd.read_csv(r'Housing.csv')

# List of variables to map
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
           'prefarea']

# Applies binary_map function to df
df[varlist] = df[varlist].apply(binary_map)

# Splits the data into training and test sets
np.random.seed(0)
```

```
df_train, df_test = train_test_split(df, train_size=0.7, test_size=0.3,
                                     random_state=np.random)

# Formats training set
# x1 : area
# x2 : bedrooms
# x3 : bathrooms
# x4 : stories
# x10: parking
num_vars = ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']
m = len(df_train)
y = df_train.values[:, 0]
x = df_train[num_vars].values[:, 2:6]  # x1 was observed to cause loss to diverge
x = np.hstack((np.ones((m, 1)), x.reshape(m, 4)))

# Retrains ML model with x2, x3, x4, x10 (x1 not usable)
theta = np.zeros(5)
theta, train_cost = gradient_descent(x, y, theta, alpha, iterations)

# Formats test set
m = len(df_test)
y_0 = df_test.values[:, 0]
x_0 = df_test[num_vars].values[:, 2:6]
x_0 = np.hstack((np.ones((m, 1)), x_0.reshape(m, 4)))

# Computes test cost
test_cost = gradient_descent(x_0, y_0, theta, alpha, iterations)[1]
print('Final value of theta =', theta)
```

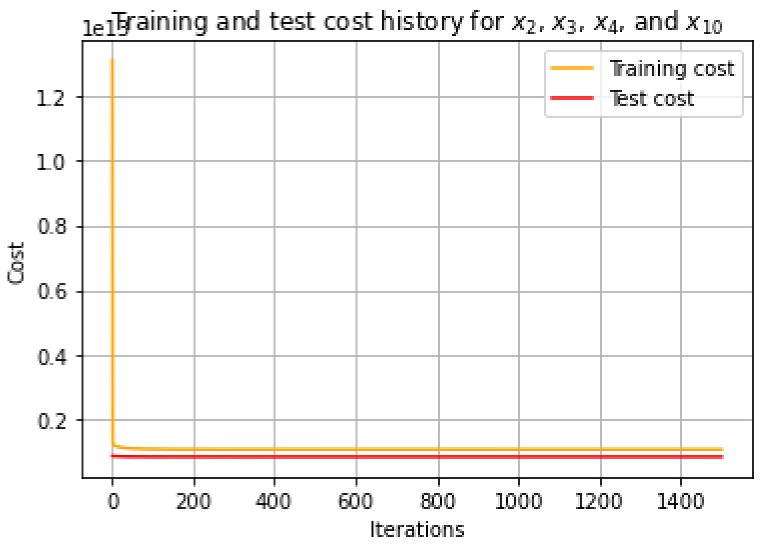
```
Final value of theta = [870489.6391051807 305563.8189222792 1203712.2817691953 561587.63
36152942
 602856.1894135593]
```

In [2]:

```
# Plots training and test cost history for x2, x3, x4, x10 (x1 not usable)
plt.figure(1)
plt.plot(np.linspace(1, iterations, iterations), train_cost, color='orange',
         label='Training cost')
plt.plot(np.linspace(1, iterations, iterations), test_cost, color='red',
         label='Test cost')
plt.rcParams['figure.figsize'] = (10, 6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Training and test cost history for $x_{2}$, $x_{3}$, $x_{4}$,'
          ' and $x_{10}$')
plt.legend()
```

Out[2]:

```
<matplotlib.legend.Legend at 0x4fe7a60>
```



In [3]:

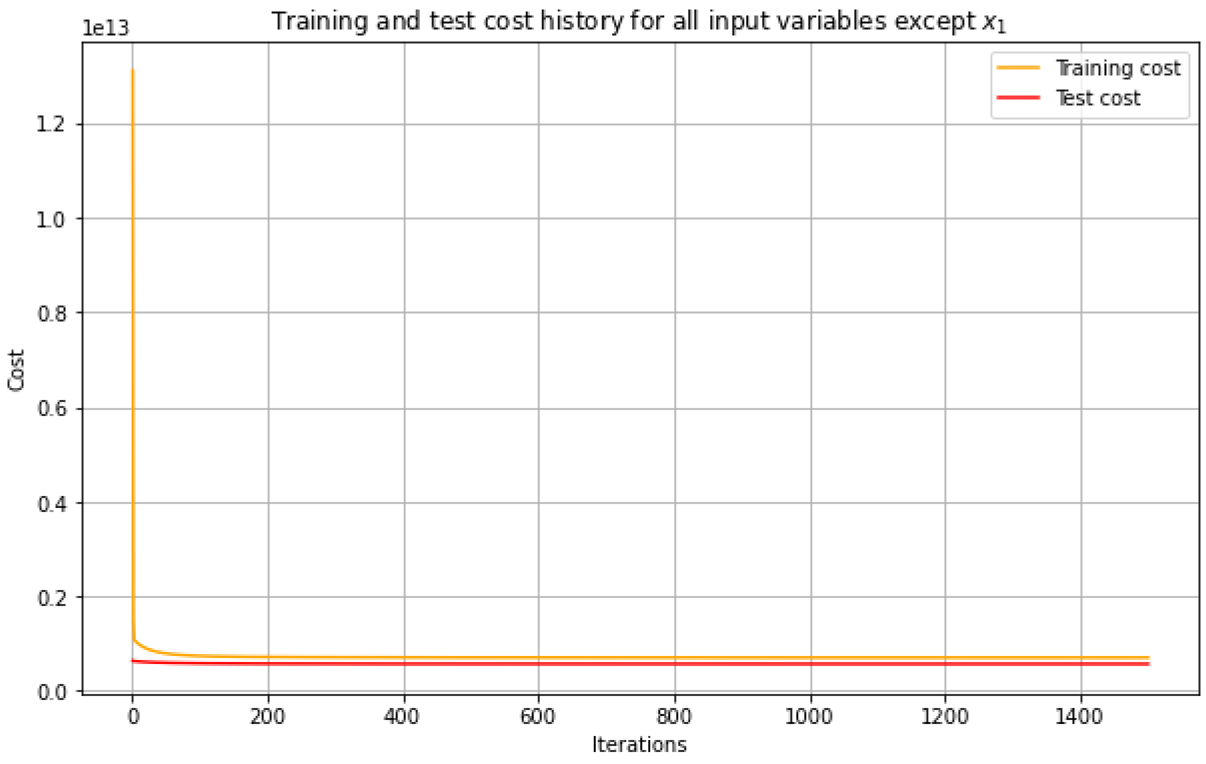
```
# Problem 1, Part b
# Formats training set
# x1 : area                x7 : basement
# x2 : bedrooms            x8 : hotwaterheating
# x3 : bathrooms           x9 : airconditioning
# x4 : stories             x10: parking
# x5 : mainroad            x11: prefarea
# x6 : guestroom
m = len(df_train)
x = df_train.values[:, 2:12]
x = np.hstack((np.ones((m, 1)), x.reshape(m, 10)))

# Retrains ML model with x2, x3, x4, x5, x6, x7, x8, x9, x10, x11 (x1 not usable)
theta = np.zeros(11)
theta, train_cost = gradient_descent(x, y, theta, alpha, iterations)

# Formats test set
m = len(df_test)
x_0 = df_test.values[:, 2:12]
x_0 = np.hstack((np.ones((m, 1)), x_0.reshape(m, 10)))

# Computes test cost
test_cost = gradient_descent(x_0, y_0, theta, alpha, iterations)[1]
print('Final value of theta =', theta)
```

```
Final value of theta = [437633.94479189994 181036.77636442526 988613.7604707936 410311.3
402010583
 756461.4261511914 538943.5229735046 265720.24743397126 1180856.4100370929
 1088780.0307373086 413179.7633270081 978459.2987293216]
```

In [4]:

```
# Plots training and test cost history for x2, x3, x4, x5, x6, x7, x8, x9, x10, x11
# (x1 not usable)
plt.figure(2)
plt.plot(np.linspace(1, iterations, iterations), train_cost, color='orange',
         label='Training cost')
plt.plot(np.linspace(1, iterations, iterations), test_cost, color='red',
         label='Test cost')
plt.rcParams['figure.figsize'] = (10, 6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Training and test cost history for all input variables except $x_{1}$')
plt.legend()
```

Out[4]:
```
<matplotlib.legend.Legend at 0xb70d6d0>
```

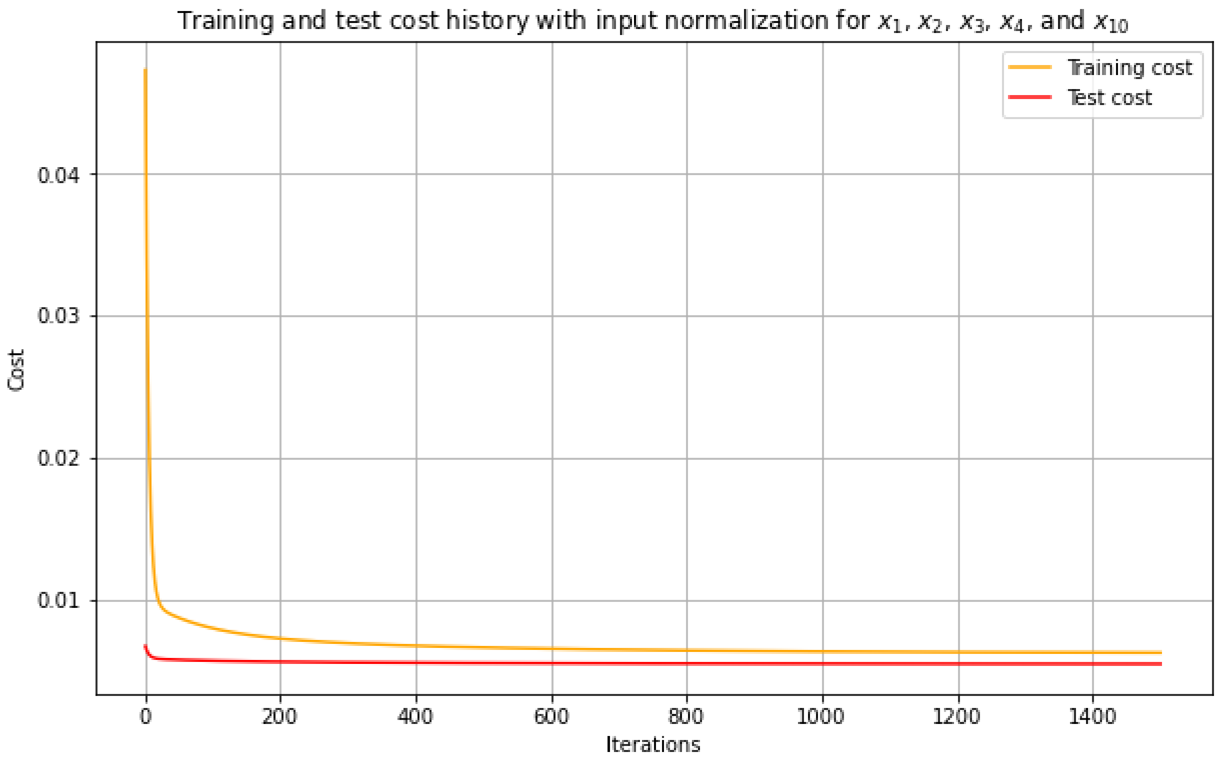


In [5]:
```
# Problem 2, Part a
# Defines MIN MAX scaler
scaler1 = MinMaxScaler()
df_train_norm = df_train[num_vars].values[:, :6]
df_test_norm = df_test[num_vars].values[:, :6]
df_train_norm = scaler1.fit_transform(df_train_norm)
df_test_norm = scaler1.fit_transform(df_test_norm)

# Formats training set
m = len(df_train_norm)
y = df_train_norm[:, 0]
x = df_train_norm[:, 1:6]
x = np.hstack((np.ones((m, 1)), x.reshape(m, 5)))

# Retrains ML model with x1, x2, x3, x4, x10
theta = np.zeros(6)
theta, train_cost = gradient_descent(x, y, theta, alpha, iterations)

# Formats test set
m = len(df_test_norm)
y_0 = df_test_norm[:, 0]
x_0 = df_test_norm[:, 1:6]
x_0 = np.hstack((np.ones((m, 1)), x_0.reshape(m, 5)))

# Computes test cost
test_cost = gradient_descent(x_0, y_0, theta, alpha, iterations)[1]
print('Final value of theta =', theta)
```

```
Final value of theta = [0.04160884 0.36839377 0.09341937 0.20038078 0.13404017 0.0969601
1]
```

In [6]:
```
# Plots training and test cost history for x1, x2, x3, x4, x10
plt.figure(3)
plt.plot(np.linspace(1, iterations, iterations), train_cost, color='orange',
         label='Training cost')
plt.plot(np.linspace(1, iterations, iterations), test_cost, color='red',
         label='Test cost')
plt.rcParams['figure.figsize'] = (10, 6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Training and test cost history with input normalization for $x_{1}$,'
          ' $x_{2}$, $x_{3}$, $x_{4}$, and $x_{10}$')
plt.legend()
```

Out[6]:
```
<matplotlib.legend.Legend at 0xb79f370>
```



In [7]:
```
# Defines standardization scaler
scaler2 = StandardScaler()
df_train_stand = df_train[num_vars].values[:, :6]
df_test_stand = df_test[num_vars].values[:, :6]
df_train_stand = scaler2.fit_transform(df_train_stand)
df_test_stand = scaler2.fit_transform(df_test_stand)

# Formats training set
m = len(df_train_stand)
y = df_train_stand[:, 0]
x = df_train_stand[:, 1:6]
x = np.hstack((np.ones((m, 1)), x.reshape(m, 5)))

# Retrains ML model with x1, x2, x3, x4, x10
theta = np.zeros(6)
theta, train_cost = gradient_descent(x, y, theta, alpha, iterations)
```

```
# Formats test set
m = len(df_test_stand)
y_0 = df_test_stand[:, 0]
x_0 = df_test_stand[:, 1:6]
x_0 = np.hstack((np.ones((m, 1)), x_0.reshape(m, 5)))

# Computes test cost
test_cost = gradient_descent(x_0, y_0, theta, alpha, iterations)[1]
print('Final value of theta =', theta)
```

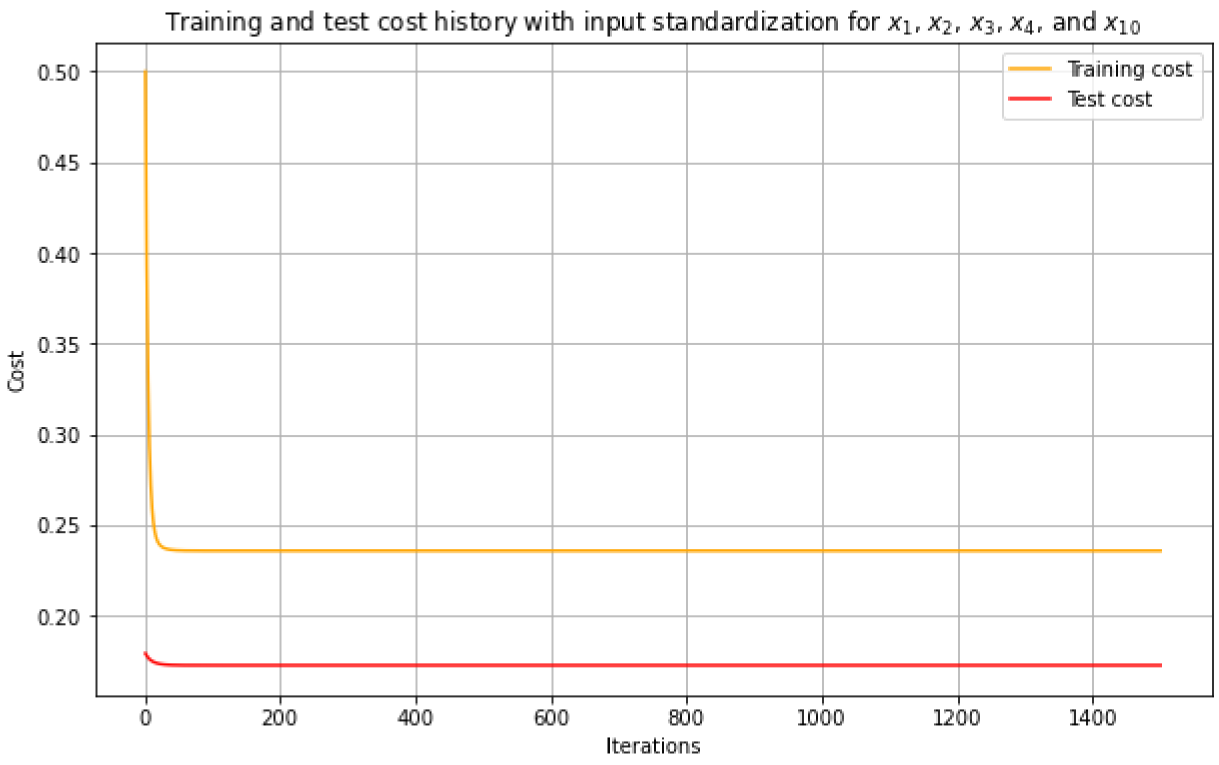
```
Final value of theta = [1.94574598e-16 3.83653304e-01 1.04343457e-01 2.98541735e-01
 2.34542828e-01 1.49757135e-01]
```

In [8]:

```
# Plots training and test cost history for x1, x2, x3, x4, x10
plt.figure(4)
plt.plot(np.linspace(1, iterations, iterations), train_cost, color='orange',
         label='Training cost')
plt.plot(np.linspace(1, iterations, iterations), test_cost, color='red',
         label='Test cost')
plt.rcParams['figure.figsize'] = (10, 6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Training and test cost history with input standardization for $x_{1}$,'
          ' $x_{2}$, $x_{3}$, $x_{4}$, and $x_{10}$')
plt.legend()
```

Out[8]:

```
<matplotlib.legend.Legend at 0xb6ff610>
```



In [9]:

```
# Problem 2, Part b
# Defines MIN MAX scaler
df_train_norm = df_train.values[:, :12]
df_test_norm = df_test.values[:, :12]
df_train_norm = scaler1.fit_transform(df_train_norm)
```

```python
df_test_norm = scaler1.fit_transform(df_test_norm)

# Formats training set
m = len(df_train_norm)
y = df_train_norm[:, 0]
x = df_train_norm[:, 1:12]
x = np.hstack((np.ones((m, 1)), x.reshape(m, 11)))

# Retrains ML model with x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11
theta = np.zeros(12)
theta, train_cost = gradient_descent(x, y, theta, alpha, iterations)

# Formats test set
m = len(df_test_norm)
y_0 = df_test_norm[:, 0]
x_0 = df_test_norm[:, 1:12]
x_0 = np.hstack((np.ones((m, 1)), x_0.reshape(m, 11)))

# Computes test cost
test_cost = gradient_descent(x_0, y_0, theta, alpha, iterations)[1]
print('Final value of theta =', theta)
```

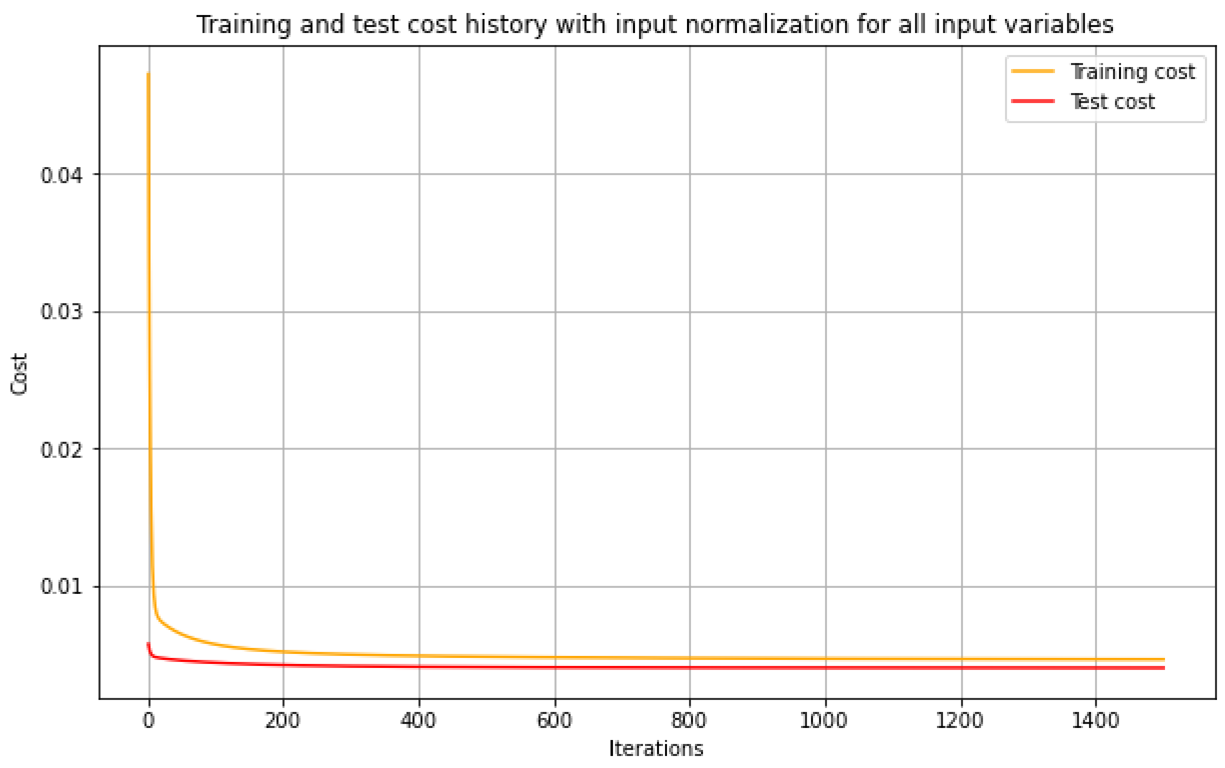
```
Final value of theta = [-0.00030068  0.24463399  0.0558249   0.166414    0.10983887  0.0
4724366
  0.03795568  0.02992177  0.10666471  0.08271132  0.07520951  0.06660711]
```

In [10]:

```python
# Plots training and test cost history for x1, x2, x3, x4, x5, x6, x7, x8, x9, x10,
# x11
plt.figure(5)
plt.plot(np.linspace(1, iterations, iterations), train_cost, color='orange',
         label='Training cost')
plt.plot(np.linspace(1, iterations, iterations), test_cost, color='red',
         label='Test cost')
plt.rcParams['figure.figsize'] = (10, 6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Training and test cost history with input normalization for all input'
          ' variables')
plt.legend()
```

Out[10]:

```
<matplotlib.legend.Legend at 0xb8f16a0>
```



In [11]:

```
# Defines standardization scaler
df_train_stand = df_train.values[:, :12]
df_test_stand = df_test.values[:, :12]
df_train_stand = scaler2.fit_transform(df_train_stand)
df_test_stand = scaler2.fit_transform(df_test_stand)

# Formats training set
m = len(df_train_stand)
y = df_train_stand[:, 0]
x = df_train_stand[:, 1:12]
x = np.hstack((np.ones((m, 1)), x.reshape(m, 11)))

# Retrains ML model with x1, x2, x3, x4, x10
theta = np.zeros(12)
theta, train_cost = gradient_descent(x, y, theta, alpha, iterations)

# Formats test set
m = len(df_test_stand)
y_0 = df_test_stand[:, 0]
x_0 = df_test_stand[:, 1:12]
x_0 = np.hstack((np.ones((m, 1)), x_0.reshape(m, 11)))

# Computes test cost
test_cost = gradient_descent(x_0, y_0, theta, alpha, iterations)[1]
print('Final value of theta =', theta)
```

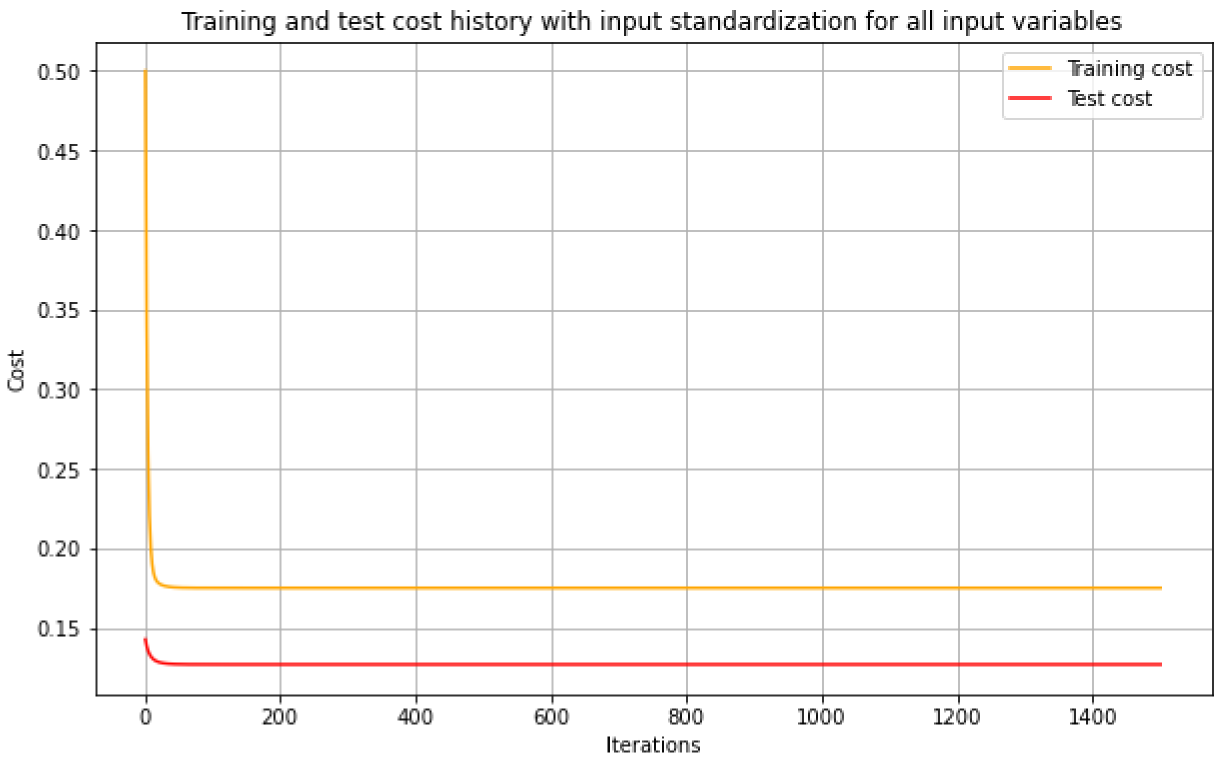
```
Final value of theta = [1.91576122e-16 2.69430270e-01 5.88474875e-02 2.48153203e-01
 1.99265962e-01 9.20205597e-02 8.25960172e-02 9.41683856e-02
 1.40206929e-01 2.32616682e-01 1.16157210e-01 1.62960286e-01]
```

In [12]:

```
# Plots training and test cost history for x1, x2, x3, x4, x5, x6, x7, x8, x9, x10,
# x11
plt.figure(6)
plt.plot(np.linspace(1, iterations, iterations), train_cost, color='orange',
         label='Training cost')
plt.plot(np.linspace(1, iterations, iterations), test_cost, color='red',
         label='Test cost')
plt.rcParams['figure.figsize'] = (10, 6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Training and test cost history with input standardization for all input'
          ' variables')
plt.legend()
```

Out[12]:

```
<matplotlib.legend.Legend at 0xbb6f430>
```



In [13]:

```
# Problem 3, Part a
# Initializes lambda to 0.001
reg_param = 0.01

# Uses MIN MAX scaler since it results in the least loss
df_train_norm = df_train[num_vars].values[:, :6]
df_test_norm = df_test[num_vars].values[:, :6]
df_train_norm = scaler1.fit_transform(df_train_norm)
df_test_norm = scaler1.fit_transform(df_test_norm)

# Formats training set
m = len(df_train_norm)
y = df_train_norm[:, 0]
x = df_train_norm[:, 1:6]
x = np.hstack((np.ones((m, 1)), x.reshape(m, 5)))

# Retrains regularized ML model with x1, x2, x3, x4, x10
theta = np.zeros(6)
theta, train_cost = gradient_descent_reg(x, y, theta, alpha, iterations, reg_param)

# Formats test set
m = len(df_test_norm)
```

```
y_0 = df_test_norm[:, 0]
x_0 = df_test_norm[:, 1:6]
x_0 = np.hstack((np.ones((m, 1)), x_0.reshape(m, 5)))

# Computes test cost
test_cost = gradient_descent(x_0, y_0, theta, alpha, iterations)[1]
print('Final value of theta =', theta)
```

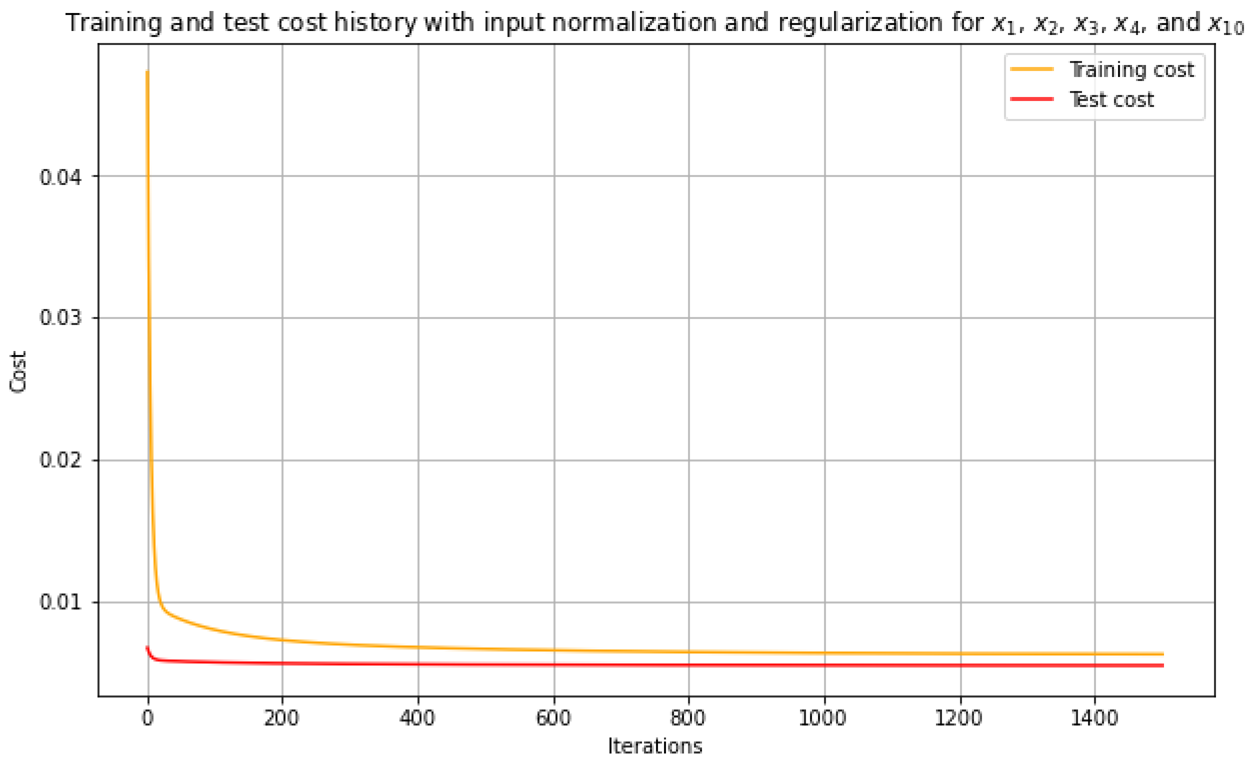
```
Final value of theta = [0.04168137 0.36808029 0.0934267  0.20031399 0.13403137 0.0969919
2]
```

In [14]:

```
# Plots training and test cost history for x1, x2, x3, x4, x10
plt.figure(7)
plt.plot(np.linspace(1, iterations, iterations), train_cost, color='orange',
         label='Training cost')
plt.plot(np.linspace(1, iterations, iterations), test_cost, color='red',
         label='Test cost')
plt.rcParams['figure.figsize'] = (10, 6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Training and test cost history with input normalization and regularization'
          ' for $x_{1}$, $x_{2}$, $x_{3}$, $x_{4}$, and $x_{10}$')
plt.legend()
```

Out[14]:

```
<matplotlib.legend.Legend at 0xba7c2e0>
```



In [15]:

```
# Problem 3, part b
# Uses MIN MAX scaler since it results in the least loss
df_train_norm = df_train.values[:, :12]
df_test_norm = df_test.values[:, :12]
df_train_norm = scaler1.fit_transform(df_train_norm)
df_test_norm = scaler1.fit_transform(df_test_norm)
```

```python
# Formats training set
m = len(df_train_norm)
y = df_train_norm[:, 0]
x = df_train_norm[:, 1:12]
x = np.hstack((np.ones((m, 1)), x.reshape(m, 11)))

# Retrains regularized ML model with x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11
theta = np.zeros(12)
theta, train_cost = gradient_descent_reg(x, y, theta, alpha, iterations, reg_param)

# Formats test set
m = len(df_test_norm)
y_0 = df_test_norm[:, 0]
x_0 = df_test_norm[:, 1:12]
x_0 = np.hstack((np.ones((m, 1)), x_0.reshape(m, 11)))

# Computes test cost
test_cost = gradient_descent(x_0, y_0, theta, alpha, iterations)[1]
print('Final value of theta =', theta)
```

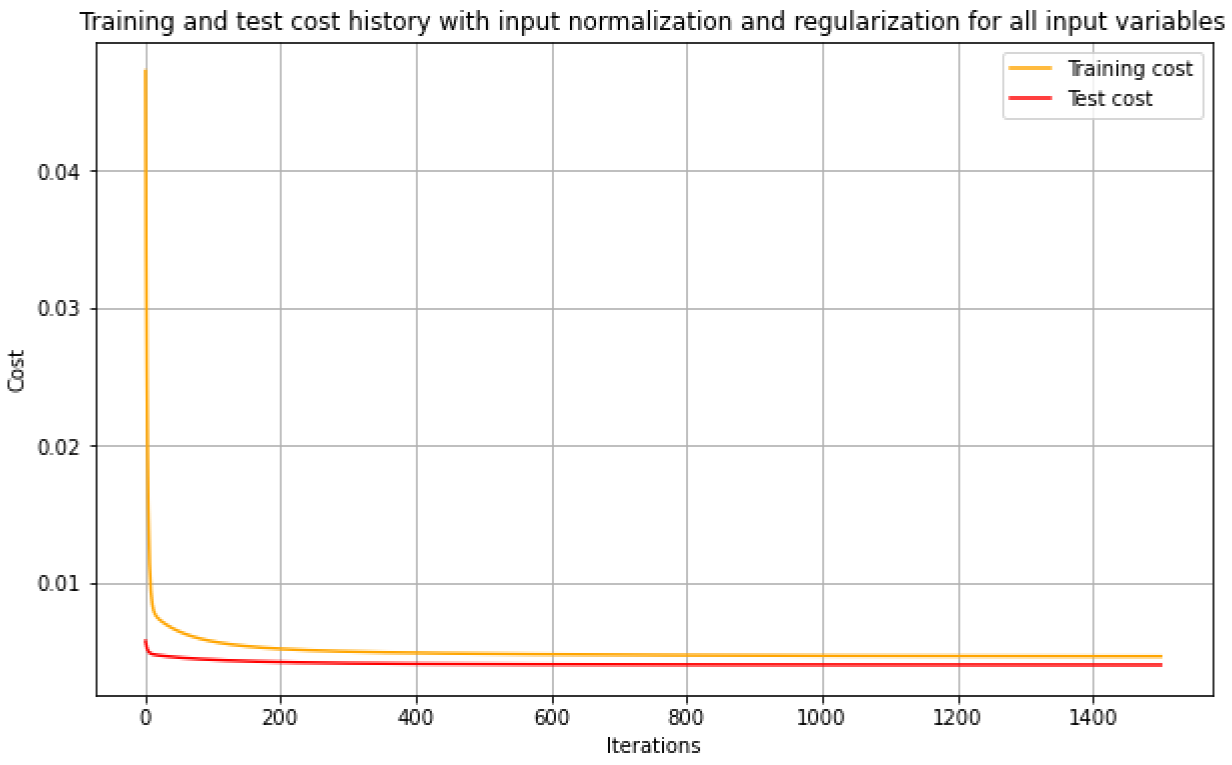
```
Final value of theta = [-0.00025713  0.24441419  0.05583525  0.166358    0.10982188  0.0
4725399
  0.03796283  0.02991754  0.1066072   0.08271691  0.07522892  0.06661762]
```

In [16]:

```python
# Plots training and test cost history for x1, x2, x3, x4, x5, x6, x7, x8, x9, x10,
# x11
plt.figure(8)
plt.plot(np.linspace(1, iterations, iterations), train_cost, color='orange',
         label='Training cost')
plt.plot(np.linspace(1, iterations, iterations), test_cost, color='red',
         label='Test cost')
plt.rcParams['figure.figsize'] = (10, 6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Training and test cost history with input normalization and regularization'
          ' for all input variables')
plt.legend()
```

Out[16]:

```
<matplotlib.legend.Legend at 0xbaef670>
```


Training and test cost history with input normalization and regularization for all input variables
Training cost
Test cost
Cost
Iterations
0.04
0.03
0.02
0.01
0
200
400
600
800
1000
1200
1400