Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU44061 Machine Learning
Lab 2

Adriana Hrabowych, 19304296

November 20, 2023

1 Introduction

In this assignment, the dataset used has the id '11-11-11'.

2 Part i

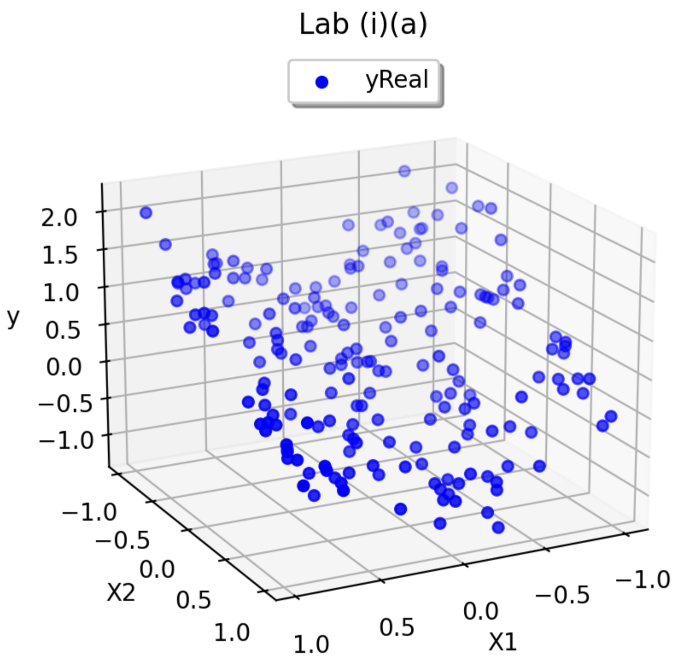


Figure 1: A 3D scatter plot graph visualizing part A of the lab.

2.1 (i)(a)

In (i)(a) the raw data is read in and organized into two arrays, one 2D array containing the two features and an 1D array containing the y valyes. Then the training data is visualized on the 3d graph seen in figure one using blue circular markers. Each data point is placed depending on the value of its two features

and the target value, with the X-axis corresponding to X1, the Y-axis corresponding to X2 and the Z-axis corresponding to the target y value.
When graphed, the training data takes on the shape of a curved plane.

2.2 (i)(b)

As seen in section (i)(a) our data lies on a curve and as such is non linear, so in order for our models to be more accurate we need to train them with additional polynomial features. In this lab we added polynomial features up to a factor of five by using sklearn's PolynomialFeatures function. This gives us a set of 21 features made up of variations on X1 and X2 which are:
$\theta_{(1-21)} = 1,\ X_1,\ X_2,\ X_1^2,\ X_1X_2,\ X_2^2,\ X_1^3,\ X_1^2X_2,\ X_1X_2^2,\ X_2^3,\ X_1^4,\ X_1^3X_2,\ X_1^2X_2^2,\ X_1X_2^3,\ X_2^4,\ X_1^5,\ X_1^4X_2,\ X_1^3X_2^2,\ X_1^2X_2^3,\ X_1X_2^4,\ X_2^5$
Now using this new set of polynomial features, four Lasso regression models were training. In Lasso, the models are linear with the addition of a cost function. The model itself is $h_\theta(x) = \theta^T x$. The objective of the model is to minimize the cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) + \frac{\lambda}{m}\sum_{j=1}^{n} |\theta_j|$$

Where C is a weight hyperparameter. In sklearn α is used instead of C, and $\alpha = 1/(2\text{C})$ The only difference between the four models trained is that they each have different C's: 1, 10, 100, and 1000. After training, the feature parameters were:

Lasso	θ1	θ2	θ3	θ4	θ5	θ6	θ7	θ8	θ9	θ10	θ11
C = 1	0	0	0	0	0	0	0	0	0	0	0
C = 10	0	0	-0.8517	0.5652	0	0	0	0	0	0	0
C = 100	0	-0.03534	-0.9898	1.0261	0	0	0	0	0	0	0
C = 1,000	0	-0.0087	-0.9829	1.0018	-0.0581	0	-0.0641	-0.0743	0	0.0257	0.0547

Figure 2: A table containing the Lasso feature parameters values for θ_1 to θ_{11} in the columns with each row representing a model with a different C value.

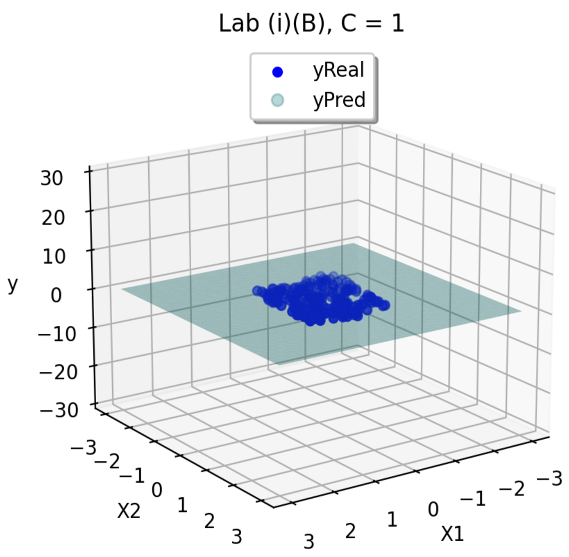
Lasso	θ12	θ13	θ14	θ15	θ16	θ17	θ18	θ19	θ20	θ21	Intercept
C = 1	0	0	0	0	0	0	0	0	0	0	0.3981
C = 10	0	0	0	0	0	0	0	0	0	0	0.1712
C = 100	0	0	0	0	0	0	0	0	0	0	0.0067
C = 1,000	0	0.074	0.1676	0.0376	0	0	0	-0.0418	0.028	0	-0.0138

Figure 3: A table containing the Lasso feature parameters values for θ_12 to θ_{21} and the intercept in the columns with each row representing a model with a different C value.

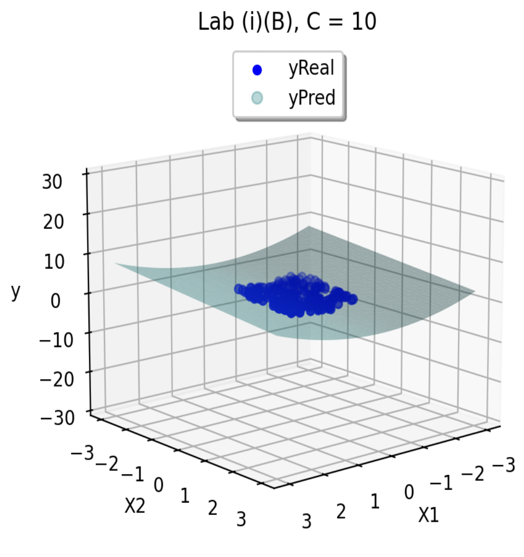
As seen in figures 2 and 3, generally as the C is raised the absolute value of each parameter also increases as the features have more influence on the predictions. The opposite is true for the intercept, which grows smaller as C increases. A lot of the feature parameters are zero as Lasso models automatically perform feature selection to eliminate the weights of less important features.

2.3 (i)(c)

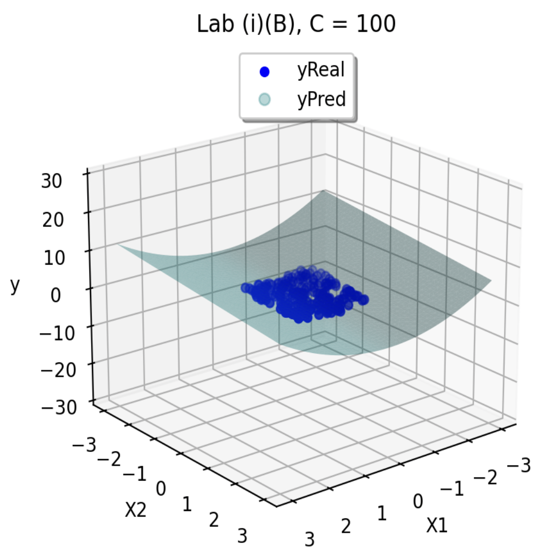
Instead of splitting our data into training and testing sections, we trained our models on all of the raw data and predict using a grid of feature values. As the original values for X_1 and X_2 were between -1 and 1, I chose to use a grid from [-3,3] as to not extend too far past the original values while still being able to see beyond the original data. The predictions are then plotted on a 3D plane.
As seen in figure 4, as C is increased the predictions begin to change from a flat linear plane (as seen when C=1) to a curved surface (seen in all other C values graphed). We originally stated that the data laid on



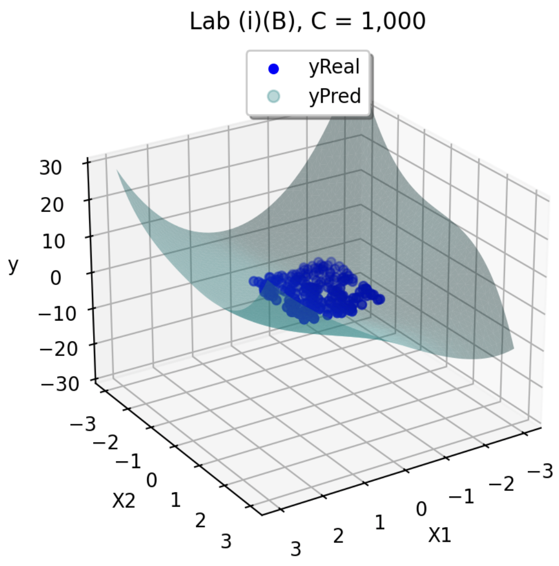
(a) Lasso model predictions with C = 1



(b) Lasso model predictions with C = 10



(c) Lasso model predictions with C = 100



(d) Lasso model predictions with C = 1,000

Figure 4: A figure with four subfigures, each containing a 3D scatter plot and plane that visualizes part ib of the lab.

a curved plane, so the models with higher C's seem to more accurately portray this. Though as the C gets much larger, the simple curve gives way to more complex structures.

2.4 (i)(d)

Underfitting is when too few features are used and as such the model cannot accurately portray the behavior of the original data. Underfit models are often incredibly inaccurate because of this. Overfitting is when too many features are used and the 'noise' in the data starts to have more influence on the predictions the model makes and as such it cannot categorize data correct.
A higher C value means the model will place a higher weight on the training data, which means that too high of a C value will cause the model to be overfit as it becomes 'overly accurate'. A lower C value puts less weight on the training data and allows more inaccuracies in predictions as it generalizes. Too low of a C can cause the lasso model to eliminate the weights of every feature parameters to 0, which causes it to be underfit. This can be seen in the graphs of figure 4, as when C=1 the surface is flat and linear and too simple to fit the data well. Whereas when C=1,000, the predicted surface becomes more complex and bumpy as it fits the data too well.

2.5 (i)(e)

Once again using the same set of polynomial features we now train four Ridge Regression models. Ridge models are linear with the addition of a penalty cost function. The objective for Ridge models is to minimize a residual sum of squares function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Where C is a complexity parameter, represented by α in sklearn, which also equals 1/2C. The only difference between the four models trained is that they each have different C's: 1, 0.1, 0.01, 0.001. After training, the feature parameters were:

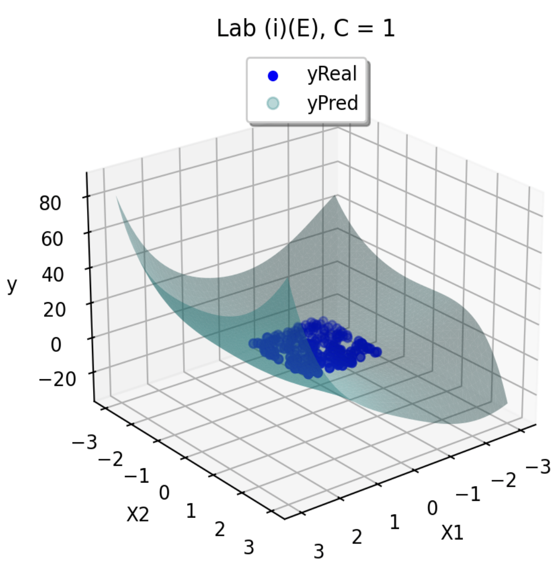
Ridge	θ1	θ2	θ3	θ4	θ5	θ6	θ7	θ8	θ9	θ10	θ11
C = 1	0	-0.0078	-0.9526	0.7613	-0.0925	-0.16	-0.1143	-0.1363	-0.0324	0.0409	0.2834
C = 0.1	0	-0.0205	-0.7368	0.532	0.0065	0.0033	-0.0404	-0.1558	0.0083	-0.1914	0.3971
C = 0.01	0	-0.0285	-0.4236	0.2354	0.01	0.0061	-0.034	-0.1263	-0.0062	-0.2221	0.2002
C = 0.001	0	-0.0122	-0.1121	0.0362	0.0048	0.0032	-0.0096	-0.0362	-0.0036	-0.0676	0.0307

Figure 5: A table containing the Ridge feature parameters values for θ_1 to θ_{11} in the columns with each row representing a model with a different C value.

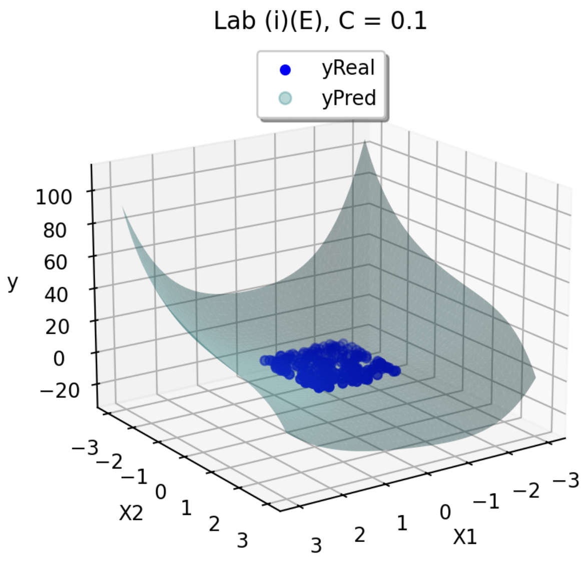
Ridge	θ12	θ13	θ14	θ15	θ16	θ17	θ18	θ19	θ20	θ21	Intercept
C = 1	0.0505	0.1742	0.1914	0.0347	0.0636	0.0463	0.0145	-0.0427	0.088	-0.0512	0.015
C = 0.1	-0.0074	0.1686	0.059	0.0012	-0.0187	-0.0576	-0.0089	-0.0267	0.0195	-0.0663	0.0702
C = 0.01	-0.0057	0.0799	0.0158	0.0015	-0.0287	-0.0707	-0.0121	-0.0608	-0.0024	-0.1479	0.2323
C = 0.001	0.0012	0.0126	0.005	0.0023	-0.0069	-0.0214	-0.0029	-0.0204	-0.0017	-0.0486	0.3672

Figure 6: A table containing the Ridge feature parameters values for θ_12 to θ_{21} and the intercept in the columns with each row representing a model with a different C value.

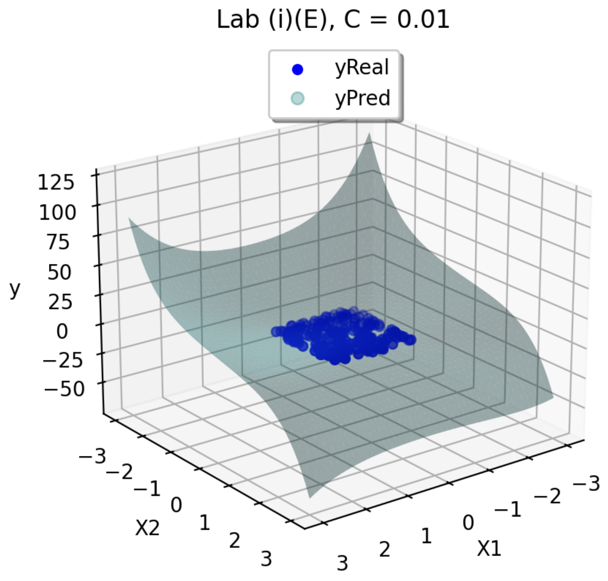
In the Ridge models as C gets bigger the feature parameters grow smaller. Unlike Lasso, as C gets larger the intercepts also get larger. The main difference between the two is the Ridge models have less parameters equal to 0 as Ridge Regression does not automatically eliminate the weight of features of lesser importance. Again the models are tested using the same grid of X values from [-3,3] and plotted using a 3D surface, as seen in figure 7. The C value in ridge models controls the strength of the l2 penalty, as C increases the variance decreases and vice versa. The C also is in charge of keeping parameters small.



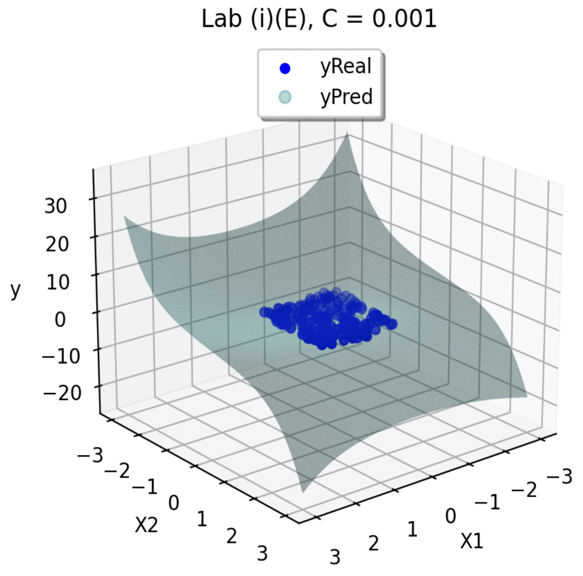
(a) Ridge model predictions with C = 1



(b) Ridge model predictions with C = 0.1

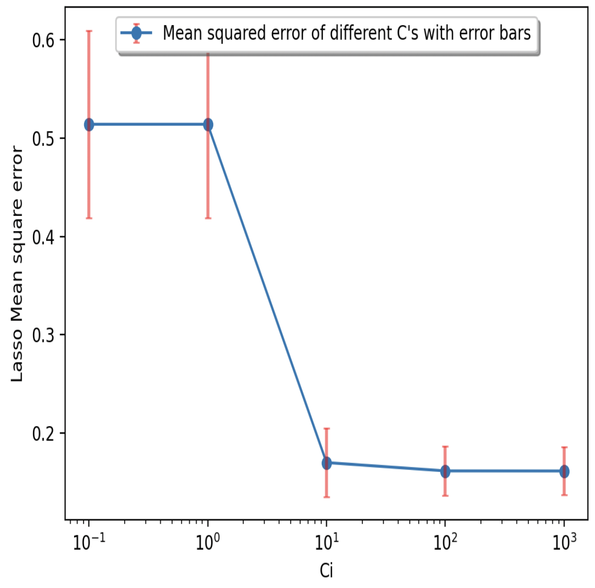


(c) Ridge model predictions with C = 0.01

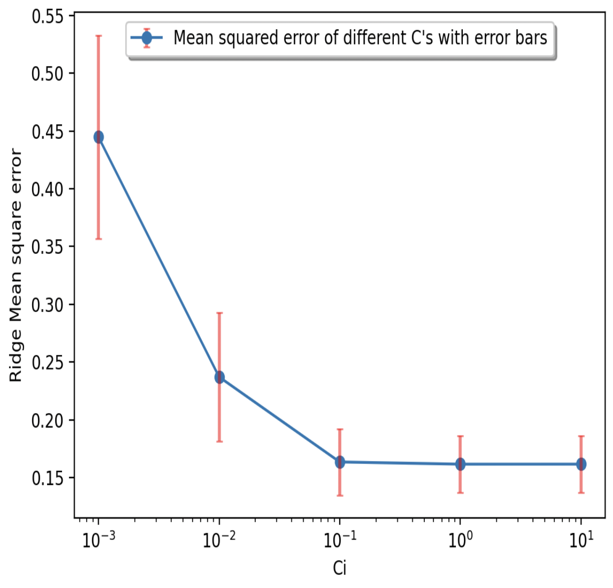


(d) Ridge model predictions with C = 0.001

Figure 7: A figure with four subfigures, each containing a 3D scatter plot and plane that visualizes part ie of the lab.



(a) Lasso Cross Validation



(b) Ridge Cross Validation

Figure 8: A figure with two subfigures, each containing a mean error bar plot that visualizes part iia of the lab.

3 Part ii

3.1 (ii)(a)

We now perform K-fold Cross Validation on Lasso models in order to see what value of C is the best one to choose. We are using a 5-fold as specified in the instructions, so we split the data into five equal parts and train the model on four of these parts and then test on the final part. We repeat this until each part has had the chance to be the testing data. With each iteration we take the mean squared error of the predictions and once the iterations are done we calculated the mean of these mean squared errors and the standard deviation.
Then we repeat this for new models with different C values, the C values I chose for this cross validation are [0.1, 1, 10, 100, 1000]. I chose these values as they are the C values I used for my models in previous parts, plus 0.1 which I added to explore going even smaller. Also, these C's extend small enough and large enough to explore both underfitting and overfitting.

3.2 (ii)(b)

Once we have the mean mean squared error and the standard deviation of these errors for each value of C, we then plot these means as points on a graph with the standard deviation creating an error bar. In figure 8.a we see the results of our Lasso Cross Validation. We can see as C gets larger, our mean and standard deviation gets smaller. It's important to note that as C increases, α in sklearn decreases. From these results it looks like a C of 10 would be the best, as any lower and our mean error is too high as we are underfit while any higher will probably overfit our data.

3.3 (ii)(c)

We then repeated the K-fold Cross Validation with Ridge Regression models and with different values of C. The values I chose for Ridge were [0.001, 0.01, 0.1, 1, 10]. I chose these values as they are the same as the ones used in my models from part ie with the addition of 10 to explore larger C's. Also because these represent a wide spread of C values that are both common and uncommon for Ridge models.
We see our Ridge cross validation results in figure 8.b. From this graph we can see that the best value of C

for our Ridge model is 0.01 or 0.1, as any larger will probably overfit our data and any smaller will underfit as seen by the large mean errors and error bars.

4 Appendix

```
# id:11-11--11
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso, Ridge
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import KFold

##Function that adds the raw data to a graph
def addRawData(graph, name, X, Y):
    graph.scatter(X[:,0], X[:,1], Y, color='b', marker='o')
    graph.set_title(name)
    graph.set_xlabel("X1")
    graph.set_ylabel("X2")
    graph.set_zlabel("y")
    if name != "Lab (i)(a)":
        graph.plot([0],[0], linestyle="none", c='teal', alpha=0.3, marker = 'o')
        graph.legend(["yReal","yPred"], loc='upper center',fancybox=True, shadow=True)
    else:
        graph.legend(["yReal"], loc='upper center',fancybox=True, shadow=True)



#(i)(a)
##Read in data
df = pd.read_csv("week3.csv")
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X=np.column_stack((X1,X2))
y=df.iloc[:,2]

##Graph Data Points
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
addRawData(ax, "Lab (i)(a)", X, y)
plt.show()

#(i)(b)
##Add polynomial features
polyX = PolynomialFeatures(5).fit_transform(np.array(X))

##Train Model
lasso1 = Lasso(alpha=0.5) #C = 1
lasso2 = Lasso(alpha=0.05) #C = 10
lasso3 = Lasso(alpha=0.005) #C = 100
lasso4 = Lasso(alpha=0.0005) #C = 1000

lasso1.fit(polyX, y)
lasso2.fit(polyX, y)
lasso3.fit(polyX, y)
lasso4.fit(polyX, y)

##Print Coefficients and Intercepts
print("Lasso C=1 Coefficients")
for i in range(21):
    print("Coefficient theta-", i+1, ": ", lasso1.coef_[i])

print("Lasso C=10 Coefficients")
for i in range(21):
    print("Coefficient theta-", i+1, ": ", lasso2.coef_[i])

print("Lasso C=100 Coefficients")
for i in range(21):
    print("Coefficient theta-", i+1, ": ", lasso3.coef_[i])

print("Lasso C=1,000 Coefficients")
for i in range(21):
    print("Coefficient theta-", i+1, ": ", lasso4.coef_[i])

print("Lasso C=1 Intercept")
print(lasso1.intercept_)
print("Lasso C=100 Intercept")
print(lasso2.intercept_)
print("Lasso C=1,000 Intercept")
print(lasso3.intercept_)
print("Lasso C=10,000 Intercept")
print(lasso4.intercept_)

#(i)(c)

xGrid = []
grid = np.linspace(-3,3)
for i in grid:
    for j in grid:
        xGrid.append([i,j])
xGrid = np.array(xGrid)
xGrid = PolynomialFeatures(5).fit_transform(xGrid)

##Predict
ypred1 = lasso1.predict(xGrid)
ypred2 = lasso2.predict(xGrid)
ypred3 = lasso3.predict(xGrid)
ypred4 = lasso4.predict(xGrid)

##Create Graphs
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
addRawData(ax, "Lab (i)(B), C = 1", X, y)
ax.plot_trisurf(xGrid[:,1], xGrid[:,2], ypred1, alpha=0.4, color='teal')
ax.set_zlim([-30, 30])
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
addRawData(ax, "Lab (i)(B), C = 10", X, y)
ax.plot_trisurf(xGrid[:,1], xGrid[:,2], ypred2, alpha=0.4, color='teal')
ax.set_zlim([-30, 30])
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
addRawData(ax, "Lab (i)(B), C = 100", X, y)
ax.plot_trisurf(xGrid[:,1], xGrid[:,2], ypred3, alpha=0.4, color='teal')
ax.set_zlim([-30, 30])
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
addRawData(ax, "Lab (i)(B), C = 1,000", X, y)
ax.plot_trisurf(xGrid[:,1], xGrid[:,2], ypred4, alpha=0.4, color='teal')
ax.set_zlim([-30, 30])
plt.show()

#(i)(e)
##Create and fit models
ridge1 = Ridge(alpha=0.5) #C = 1
ridge2 = Ridge(alpha=5.0) #C = 0.1
ridge3 = Ridge(alpha=50.0) #C = 0.01
ridge4 = Ridge(alpha=500.0) #C = 0.001

```

```
ridge1.fit(polyX, y)
ridge2.fit(polyX, y)
ridge3.fit(polyX, y)
ridge4.fit(polyX, y)

##Print Coefficients and Intercepts
print("Ridge C=1 Coefficients")
for i in range(21):
    print("Coefficient theta-", i+1, ": ", ridge1.coef_[i])

print("Ridge C=0.1 Coefficients")
for i in range(21):
    print("Coefficient theta-", i+1, ": ", ridge2.coef_[i])

print("Ridge C=0.01 Coefficients")
for i in range(21):
    print("Coefficient theta-", i+1, ": ", ridge3.coef_[i])

print("Ridge C=0.001 Coefficients")
for i in range(21):
    print("Coefficient theta-", i+1, ": ", ridge4.coef_[i])

print("Ridge C=1 Intercept")
print(ridge1.intercept_)
print("Ridge C=0.1 Intercept")
print(ridge2.intercept_)
print("Ridge C=0.01 Intercept")
print(ridge3.intercept_)
print("Ridge C=0.001 Intercept")
print(ridge4.intercept_)

##Predict
ypred1 = ridge1.predict(xGrid)
ypred2 = ridge2.predict(xGrid)
ypred3 = ridge3.predict(xGrid)
ypred4 = ridge4.predict(xGrid)

##Create Graphs
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
addRawData(ax, "Lab (i)(E), C = 1", X, y)
ax.plot_trisurf(xGrid[:,1], xGrid[:,2], ypred1, alpha=0.4, color='teal')
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
addRawData(ax, "Lab (i)(E), C = 0.1", X, y)
ax.plot_trisurf(xGrid[:,1], xGrid[:,2], ypred2, alpha=0.4, color='teal')
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
addRawData(ax, "Lab (i)(E), C = 0.01", X, y)
ax.plot_trisurf(xGrid[:,1], xGrid[:,2], ypred3, alpha=0.4, color='teal')
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
addRawData(ax, "Lab (i)(E), C = 0.001", X, y)
ax.plot_trisurf(xGrid[:,1], xGrid[:,2], ypred4, alpha=0.4, color='teal')
plt.show()

#(ii)(a)

mean_error=[]; std_error=[]
Ci_range = [0.1, 1, 10, 100, 1000]
for Ci in Ci_range:
    model = Lasso(alpha=1/(2*Ci))
    temp=[]
    kf = KFold(n_splits=5)
    for train, test in kf.split(polyX):
        model.fit(X[train], y[train])
        ypred = model.predict(X[test])
        temp.append(mean_squared_error(y[test], ypred))
    mean_error.append(np.array(temp).mean())
    std_error.append(np.array(temp).std())

markers, caps, bars = plt.errorbar(Ci_range, mean_error, barsabove=True, yerr=std_error,
    ecolor='r', capsize=2, elinewidth=2, fmt="-o")
[bar.set_alpha(0.5) for bar in bars]
[cap.set_alpha(0.5) for cap in caps]

plt.xlabel('Ci')
plt.legend(["Mean squared error of different C's with error bars"], loc='upper center',
    fancybox=True, shadow=True)
plt.ylabel('Lasso Mean square error')
plt.xscale('log')
plt.show()

#(ii)(c)

mean_error=[]; std_error=[]
Ci_range = [0.001, 0.01, 0.1, 1, 10]
for Ci in Ci_range:
    model = Ridge(alpha=1/(2*Ci))
    temp=[]
    kf = KFold(n_splits=5)
    for train, test in kf.split(polyX):
        model.fit(polyX[train], y[train])
        ypred = model.predict(polyX[test])
        temp.append(mean_squared_error(y[test], ypred))
    mean_error.append(np.array(temp).mean())
    std_error.append(np.array(temp).std())

markers, caps, bars = plt.errorbar(Ci_range, mean_error, barsabove=True, yerr=std_error,
    ecolor='r', capsize=2, elinewidth=2, fmt="-o")
[bar.set_alpha(0.5) for bar in bars]
[cap.set_alpha(0.5) for cap in caps]

plt.xlabel('Ci')
plt.legend(["Mean squared error of different C's with error bars"], loc='upper center',
    fancybox=True, shadow=True)
plt.ylabel('Ridge Mean square error')
plt.xscale('log')
plt.show()
```