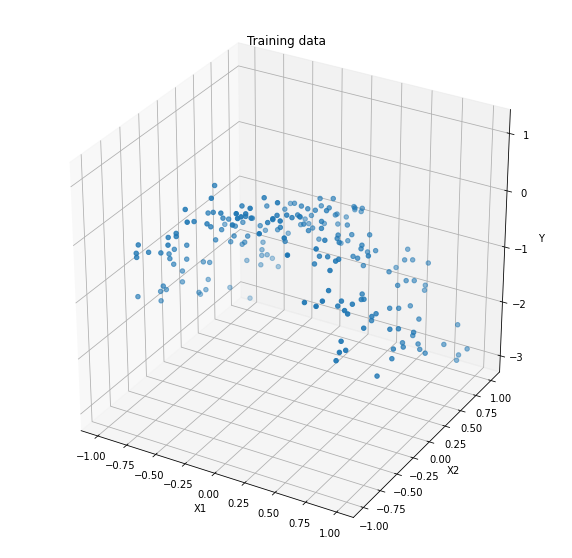
# Report

Q1

I use the plt 3d plot to generate the plot below:



As we can see the figure above. It shows us that the data lies on a curve.

b

I use the sklearn PolynomialFeatures function to add extra polynomial features.

The features are list below:

1

x0

x1

x0^2

x0 x1

x1^2

x0^3

x0^2 x1

x0 x1^2

x1^3

x0^4

x0^3 x1

x0^2 x1^2

x0 x1^3

x1^4

x0^5

x0^4 x1

x0^3 x1^2

x0^2 x1^3

x0 x1^4

x1^5

Then I train Lasso regression models with these polynomial features for a large range of values of C. [1, 10, 1000, 10000]. For each c value. The mean squared error is calculates to evaluate diﬀerent models.

The following values are model parameters and MSE value in different values of C.

C = 1

mse = 0.7251

theta = [ 0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0.

-0. -0. -0.]

theta0 = -0.6674421439324679

C = 10

mse = 0.0818

theta = [ 0. -0. -0.86594124 -1.50202448 -0. -0. -0. -0. -0. -0. -0. -0. -0. 0. -0. -0. -0. -0. -0. -0. -0. ]

theta0 = -0.14973970090816424

C = 1000

mse = 0.0429

theta = [ 0. -0.03061585 -1.05258724 -2.06786679 -0.07360959 -0. -0. -0. 0. 0. 0. -0. -0. 0.11961047 -0.06866554 0.07756016 0.0085111 -0.00577164 0. 0.01181523 0. ]

theta0 = 0.05742499257444211

C = 10000

mse = 0.0418

theta = [ 0. 0.00395004 -1.03492648 -2.1681133 -0.13016784 0.07198767 -0.34503859 -0.31927843 0.10378258 0.01339742 0.10331555 -0. 0.03492085 0.18171726 -0.17472084 0.48142321 0.36697515 -0.39145778 0.01955836 0.19654193 0. ]

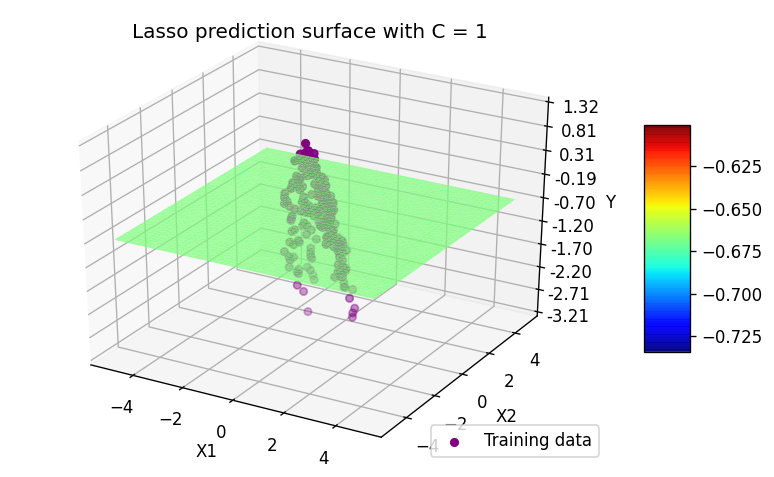
theta0 = 0.061567465046199366

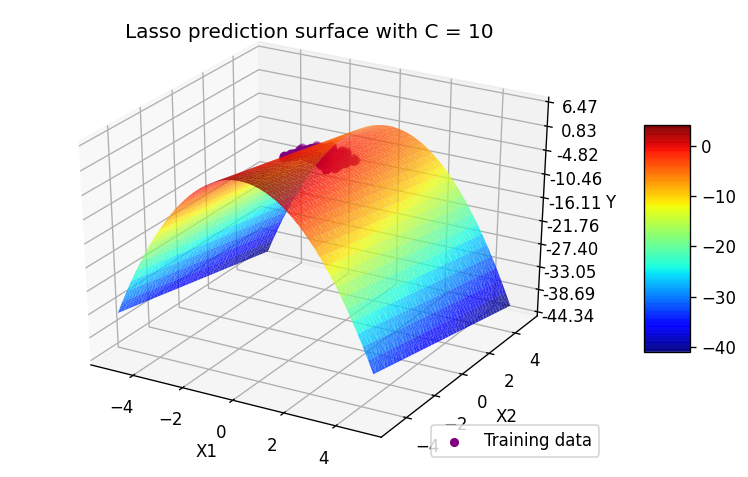
From the result above we can see that:

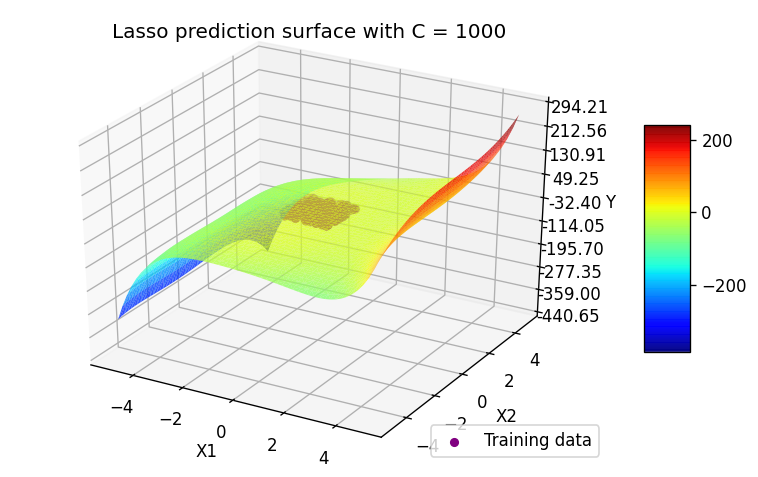
The L1 penalty of Lasso regression model makes many elements of theta are 0. The bigger C is the less elements of theta parameter are 0 and the small MSE value which means the model performance is better. We can also infer that the feature x0 is important. Since the polynomial features that involve x1 parameters are small. For example, the x1^5 are always 0.

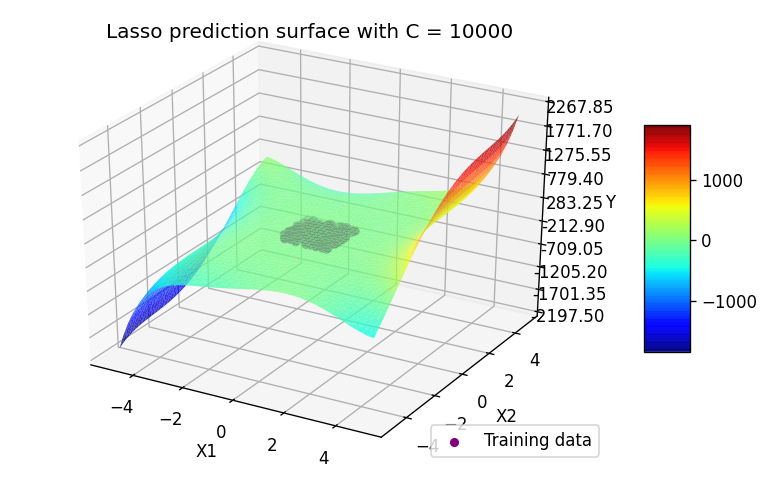
c

The following plots C {1, 10, 1000, 10000}









As the plot shows that the prediction surface gets more complex with the C value increase.

d

The under-ﬁtting is that the model is too simple to learn the underlying feature of data which will lead to poor predictions. As we can see in our training data: in plot 1 using a ﬂat prediction surface for this data is an example of under-ﬁtting.

The over-ﬁtting is that the model is learn too much about the training data even small variations of training data which will also lead to leading to poor predictions. Like the C=10000 plot the shape is too complex.

It is a great option to fine-turn the hyperparameter C to control the high-degree features of our training data to use in our model. In that case, we can control the model complex which avoid under or over-fitting.

e

In Ridge Regression we use C [1e-7, 1e-5, 1e-1, 1], and repeat the same steps as for questions above:

C = 1e-07

mse = 0.7251

theta = [ 0.00000000e+00 -1.09647910e-06 -1.20020623e-05 -6.98954233e-06

-4.02898578e-07 -5.37844106e-07 -8.83936363e-07 -4.29959280e-06

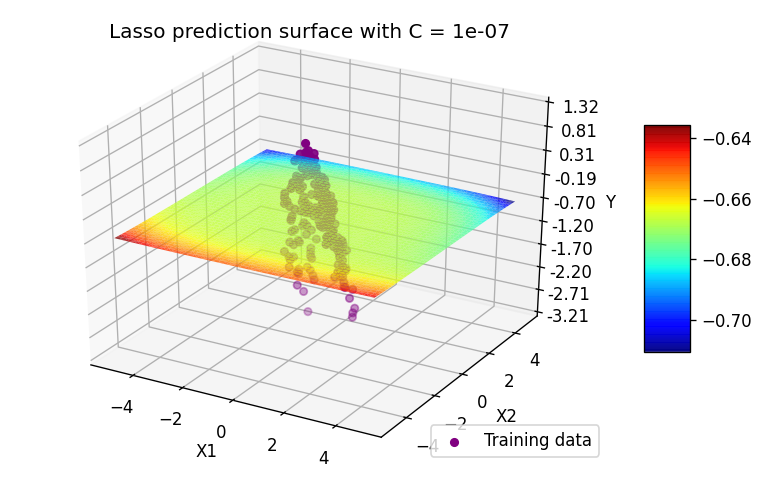
-4.26107135e-07 -7.55798857e-06 -6.12915606e-06 -3.90163144e-07

-2.27554189e-06 -2.55428203e-07 -6.41729852e-07 -7.29220833e-07

-2.66254735e-06 -3.57286176e-07 -2.85247908e-06 -2.04095069e-07

-5.64975155e-06]

theta0 = -0.6674378084360448



C = 1e-05

mse = 0.7233

theta = [ 0.00000000e+00 -1.09345848e-04 -1.19754595e-03 -6.98573268e-04

-4.00578451e-05 -5.36716469e-05 -8.81980215e-05 -4.28918474e-04

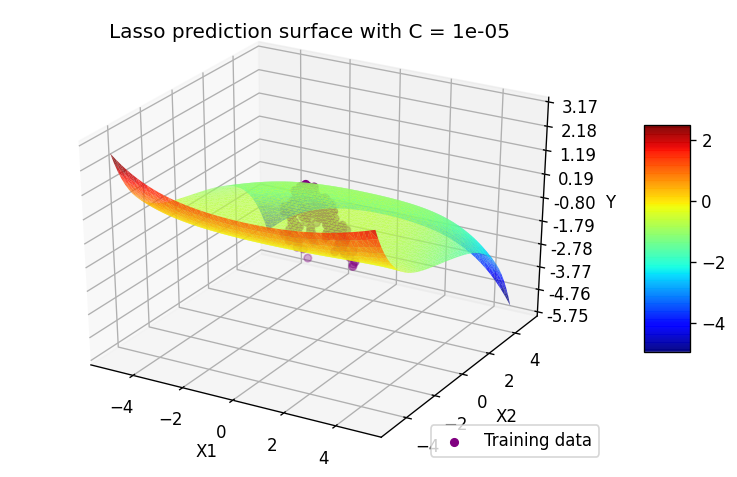
-4.25544411e-05 -7.54003331e-04 -6.12536530e-04 -3.88345138e-05

-2.27403792e-04 -2.53345145e-05 -6.40678625e-05 -7.27660718e-05

-2.65589049e-04 -3.57091641e-05 -2.84527212e-04 -2.04116146e-05

-5.63575022e-04]

theta0 = -0.6670088978809444



C = 0.1

mse = 0.0600

theta = [ 0.00000000e+00 -1.35443455e-02 -7.72295230e-01 -1.06042172e+00

-1.73953142e-02 2.36536033e-02 3.49005858e-04 -1.13203264e-01

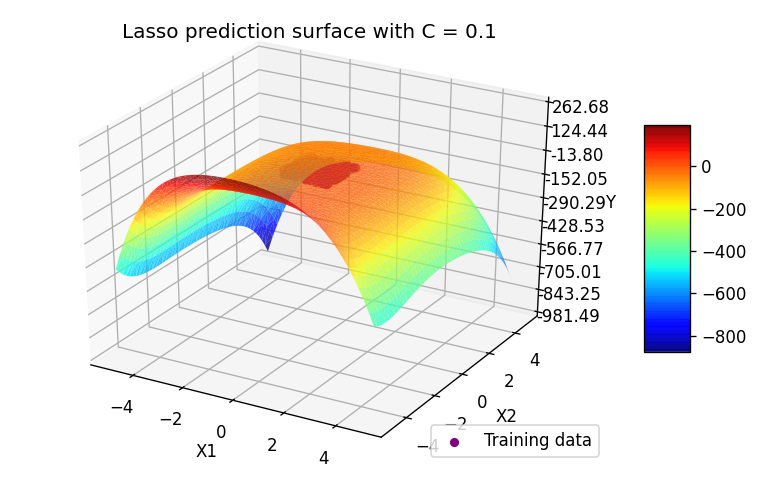
2.22749144e-03 -2.15522438e-01 -7.26509218e-01 -1.01561510e-03

-2.40102839e-01 5.95142371e-02 -2.46576227e-02 3.28310760e-02

-1.95283989e-03 -1.92436108e-02 1.28663919e-02 2.99070616e-04

-6.03935024e-02]

theta0 = -0.12504155175548615



C = 1

mse = 0.0441

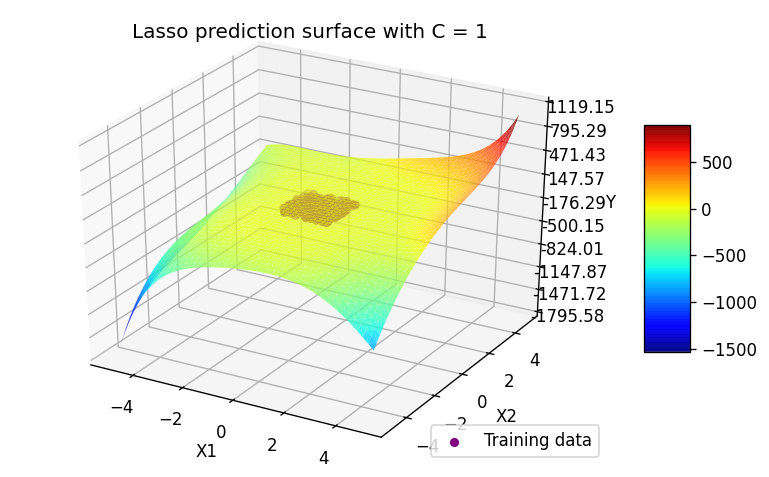
theta = [ 0. -0.03430445 -0.98140241 -1.67284647 -0.08498967 0.05909102

-0.05934594 -0.17358945 0.02224946 -0.10741457 -0.36843846 -0.012031

-0.12367714 0.13116568 -0.10021716 0.16401253 0.15727365 -0.09720448

0.08616842 0.08903423 0.0390254 ]

theta0 = -0.0009510319698949887



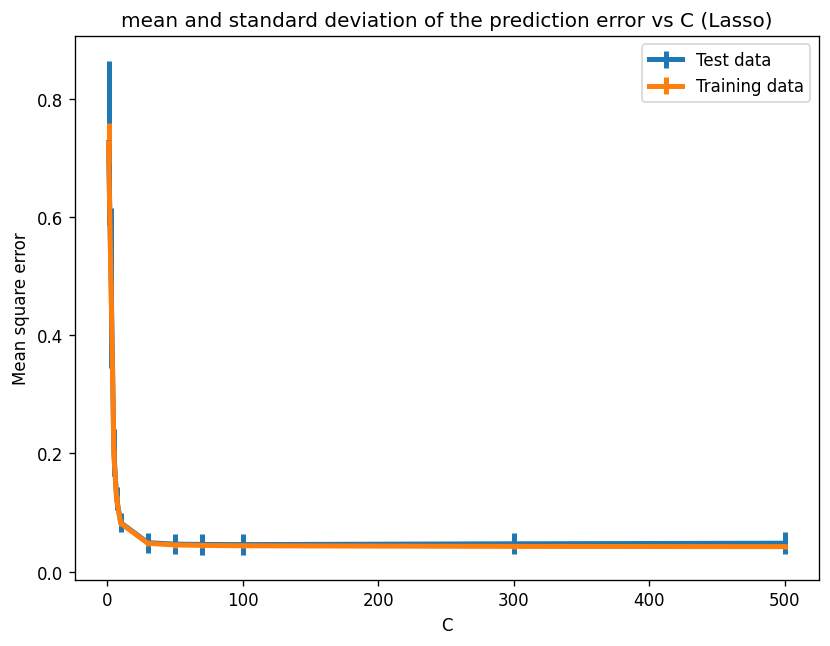
The L2 penalty in Ridge regression model that makes parameters small values. As we can see from the result theta and plots above: the hyperparameter C able to control L2 penalty: the bigger the C is the smaller the penalty is and more complex model is.

Compare the impact on the model parameters of changing C with Lasso Regression and with Ridge Regression.: In Lasso model, we can make some features to zero by changing C. In Ridge model, we only can decrease or increase the parameter values.

Q2

a

The C value range I choose is [1, 3, 5, 7, 10, 30, 50, 70, 100, 300, 500]. The reason I choose this range is that: It increase the C value in small steps and then increase in large steps which will allow us to observe the prediction error value change in early stage.



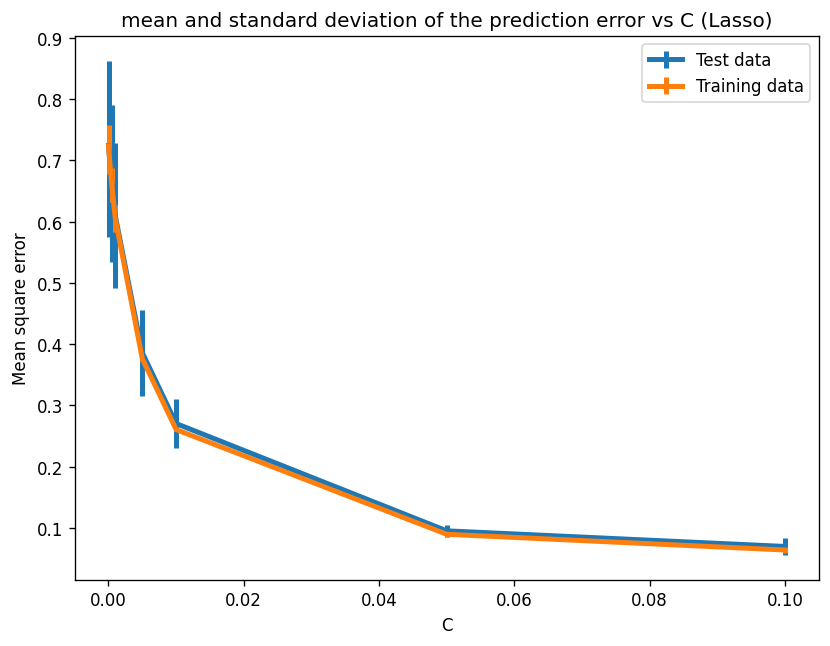
b

Based on the cross-validation data what value of C, we can see that the best result are obtained for C ≥ 30, with smallest MSE and standard deviation In order to prevent over-ﬁtting we can choose C=30 or 50. In common, the more simple the model is the model is better. So I recommend use C=30.

c

For Ridge Regression model, the C value range I choose is [1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1]. The reason I choose this range is that: It increase the C value in small steps and then increase in large steps which will allow us to observe the prediction error value change in early stage

As we can see the plot below:



We can see that the best result is obtained for C = 0.1, with smallest MSE and standard deviation But In order to prevent over-ﬁtting we can choose C=0.1 or 0.05 In common, the more simple the model is the model is better. So I recommend use C= 0.05.

# Appendix

# -\*- coding: utf-8 -\*-

"""cs7cs4.ipynb

# Q1

## a

"""

# Commented out IPython magic to ensure Python compatibility.

import numpy as np

import pandas as pd

# %matplotlib inline

df = pd.read\_csv('week3.txt',comment="#", header=None)

df

X = df.iloc[:,:2]

X

y = df.iloc[:,2]

y

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 10))

ax = fig.add\_subplot(111, projection ='3d')

ax.scatter(X[0], X[1] , y)

ax.set(title='Training data', xlabel='X1', ylabel='X2', zlabel='Y')

plt.show()

"""## b"""

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import Lasso

from sklearn.metrics import mean\_squared\_error

poly = PolynomialFeatures(degree=5)

X5poly = poly.fit\_transform(X)

for f in poly.get\_feature\_names\_out():

print(f)

C\_range = [1, 10, 1000, 10000]

for C in C\_range:

model = Lasso(alpha=1/(2\*C)).fit(X5poly, y)

theta = model.coef\_

theta0 = model.intercept\_

y\_pred = model.predict(X5poly)

j = mean\_squared\_error(y, y\_pred)

print(f"C = {C}\nmse = {j:.4f}\n")

print(f'theta = {theta}\ntheta0 = {theta0}\n')

"""## c"""

Xtest = []

grid = np.linspace(-5,5)

for i in grid:

for j in grid:

Xtest.append([i,j])

Xtest = np.array(Xtest)

Xtest = PolynomialFeatures(5).fit\_transform(Xtest)

from matplotlib import cm

from matplotlib.ticker import LinearLocator, FormatStrFormatter

C\_range = [1, 10, 1000, 10000]

for C in C\_range:

model = Lasso(alpha=1/(2\*C))

model.fit(X5poly, y)

Z = model.predict(Xtest)

fig = plt.figure(figsize=(8, 5), dpi=120)

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(X[0], X[1], y, c='purple', label="Training data")

surf = ax.plot\_trisurf(Xtest[:,1], Xtest[:,2], Z,

cmap=cm.jet, alpha=0.8, linewidth=0, antialiased=True)

ax.set\_title(f'Lasso prediction surface with C = {C}')

# Customize the z axis.

ax.zaxis.set\_major\_locator(LinearLocator(10))

# A StrMethodFormatter is used automatically

ax.zaxis.set\_major\_formatter(FormatStrFormatter('%.02f'))

# Add a color bar which maps values to colors.

fig.colorbar(surf, shrink=0.5, aspect=5)

ax.set(xlabel='X1', ylabel='X2', zlabel='Y')

ax.legend(loc='lower right')

plt.show()

"""## e"""

# (i)(e)

from sklearn.linear\_model import Ridge

C\_range = [1e-7, 1e-5, 1e-1, 1]

for C in C\_range:

model = Ridge(alpha=1/(2\*C)).fit(X5poly, y)

theta = model.coef\_

theta0 = model.intercept\_

y\_pred = model.predict(X5poly)

j = mean\_squared\_error(y, y\_pred)

print(f"C = {C}\nmse = {j:.4f}\n")

print(f'theta = {theta}\ntheta0 = {theta0}\n')

model.fit(X5poly, y)

Z = model.predict(Xtest)

fig = plt.figure(figsize=(8, 5), dpi=120)

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(X[0], X[1], y, c='purple', label="Training data")

surf = ax.plot\_trisurf(Xtest[:,1], Xtest[:,2], Z,

cmap=cm.jet, alpha=0.8, linewidth=0, antialiased=True)

ax.set\_title(f'Lasso prediction surface with C = {C}')

# Customize the z axis.

ax.zaxis.set\_major\_locator(LinearLocator(10))

# A StrMethodFormatter is used automatically

ax.zaxis.set\_major\_formatter(FormatStrFormatter('%.02f'))

# Add a color bar which maps values to colors.

fig.colorbar(surf, shrink=0.5, aspect=5)

ax.set(xlabel='X1', ylabel='X2', zlabel='Y')

ax.legend(loc='lower right')

plt.show()

"""# Q2

## a

"""

from sklearn.model\_selection import KFold

from sklearn import linear\_model

from sklearn.metrics import mean\_squared\_error

test\_std = []

test\_mean = []

train\_std = []

train\_mean = []

plt.figure(figsize=(8, 6), dpi=120)

C\_range = [1, 3, 5, 7, 10, 30, 50, 70, 100, 300, 500]

kfcv = KFold(n\_splits=5)

for C in C\_range:

model = Lasso(alpha=1/(2\*C))

test\_mse = []

train\_mse = []

for train, test in kfcv.split(X5poly):

model.fit(X5poly[train], y[train])

y\_pred\_test = model.predict(X5poly[test])

y\_pred\_train = model.predict(X5poly[train])

test\_mse.append(mean\_squared\_error(y[test], y\_pred\_test))

train\_mse.append(mean\_squared\_error(y[train], y\_pred\_train))

test\_mean .append(np.mean(test\_mse))

test\_std.append(np.std(test\_mse))

train\_mean.append(np.mean(train\_mse))

train\_std.append(np.std(train\_mse))

plt.errorbar(C\_range, test\_mean, yerr=test\_std, linewidth=3, label="Test data")

plt.errorbar(C\_range, train\_mean, yerr=train\_std, linewidth=3, label="Training data")

plt.title("mean and standard deviation of the prediction error vs C (Lasso)")

plt.xlabel('C')

plt.ylabel("Mean square error")

plt.legend()

plt.show()

"""## c"""

test\_std = []

test\_mean = []

train\_std = []

train\_mean = []

plt.figure(figsize=(8, 6), dpi=120)

C\_range = [1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1]

kfcv = KFold(n\_splits=5)

for C in C\_range:

model = Ridge(alpha=1/(2\*C))

test\_mse = []

train\_mse = []

for train, test in kfcv.split(X5poly):

model.fit(X5poly[train], y[train])

y\_pred\_test = model.predict(X5poly[test])

y\_pred\_train = model.predict(X5poly[train])

test\_mse.append(mean\_squared\_error(y[test], y\_pred\_test))

train\_mse.append(mean\_squared\_error(y[train], y\_pred\_train))

test\_mean .append(np.mean(test\_mse))

test\_std.append(np.std(test\_mse))

train\_mean.append(np.mean(train\_mse))

train\_std.append(np.std(train\_mse))

plt.errorbar(C\_range, test\_mean, yerr=test\_std, linewidth=3, label="Test data")

plt.errorbar(C\_range, train\_mean, yerr=train\_std, linewidth=3, label="Training data")

plt.title("mean and standard deviation of the prediction error vs C (Lasso)")

plt.xlabel('C')

plt.ylabel("Mean square error")

plt.legend()

plt.show()