Homework Three

Kyle Kazemini

October 20, 2020

Question 1

(a)

```
import numpy as np
import scipy as sp

x = np.genfromtxt('http://www.cs.utah.edu/~jeffp/teaching/FoDA/x.csv')
y = np.genfromtxt('http://www.cs.utah.edu/~jeffp/teaching/FoDA/y.csv')

(a, b) = np.polyfit(x, y, 1)
print(a, b)
```

From the given code, the coefficients for the model are 515.1222963490035 and -1095.6026286820393.

For $x = 4 : y = 515.122296349003(4) - 1095.6026286820393 = 964.886556714$

For $x = 8.5 : y = 515.122296349003(8.5) - 1095.6026286820393 = 3282.93689028$

(b)

```
import numpy as np
import scipy as sp

x = np.genfromtxt('http://www.cs.utah.edu/~jeffp/teaching/FoDA/x.csv')
y = np.genfromtxt('http://www.cs.utah.edu/~jeffp/teaching/FoDA/y.csv')

(a, b) = np.polyfit(x, y, 1)
print(a, b)

xTrain = np.resize(x, 80)
yTrain = np.resize(y, 80)

xTest = x[-20:]
yTest = y[-20:]

(c, d) = np.polyfit(xTrain, yTrain, 1)
print(c, d)
```

From the given code, the coefficients for the model are 492.83938094054713 and -991.1378015106926.

For $x = 4 : y = 492.83938094054713(4) - 991.1378015106926 = 980.219722251$

For $x = 8.5 : y = 492.83938094054713(8.5) - 991.1378015106926 = 3197.99693648$

(c)

```python
xTrain = np.resize(x, 80)
yTrain = np.resize(y, 80)

xTest = x[-20:]
yTest = y[-20:]

(c, d) = np.polyfit(xTrain, yTrain, 1)
print(c, d)

print(x[0:20] - yTest)
print(LA.norm(x[0:20] - yTest))

print(xTrain[0:20] - yTest)
print(LA.norm(xTrain[0:20] - yTest))
```

[-1872.97566844 -2124.20042923 -2806.38564739 -685.21763991 -612.44185309 -311.45530061 -2073.90359254 -6329.50685967 -1426.50893435 -1752.68576377 -3341.71890547 -2172.88332434 -1333.15364208 -2482.68733743 -1677.87229317 -1953.870498 -1989.81298483 -1408.72616297 -1704.86156841 -1568.61698997]

10396.231606675967

[-1872.97566844 -2124.20042923 -2806.38564739 -685.21763991 -612.44185309 -311.45530061 -2073.90359254 -6329.50685967 -1426.50893435 -1752.68576377 -3341.71890547 -2172.88332434 -1333.15364208 -2482.68733743 -1677.87229317 -1953.870498 -1989.81298483 -1408.72616297 -1704.86156841 -1568.61698997]

10396.231606675967

(d)

The first 3 rows of the matrix $\tilde{X}_3$ are:

$$\begin{bmatrix} 1 & 3.64641412 & 13.2963359345 & 48.483947096 \\ 1 & 9.76756879 & 95.4054000674 & 931.878808096 \\ 1 & 2.72543008 & 7.42796912097 & 20.2444104756 \end{bmatrix}$$

```python
coef = np.polyfit(xTrain, yTrain, 3)
print(coef)
```

[2.20876179 22.49361866 42.98550892 175.37688997]
Thus, the model is of the form $2.20876179 + 22.49361866x + 42.98550892x^2 + 175.37688997x^3$.

```python
print(xTrain[0:4] - coef)
print(LA.norm(xTrain[0:4] - coef))

print(xTest[0:4] - coef)
print(LA.norm(xTest[0:4] - coef))
```

[1.43765233 -12.72604987 -40.26007885 -171.33807389]
176.46990877821662

[3.98029592 -16.25780541 -35.44440414 -172.10644721]
176.51372169996213

Question 2

(a)

The span of the columns of X is all of $\mathbb{R}^n$ because the columns of X are linearly independent.

(b)

X is a square matrix because the number of rows n equals the number of columns n. By definition, the rank of a matrix is equal to the column rank. Since (a) shows that X has full column rank, X also has full rank. Thus, X is an invertible matrix.

(c)

Since Reginald chose the correct $\hat{\alpha}$, from the textbook we know this to be

$$\hat{\alpha} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T y$$

$\hat{\alpha}$ is an n dimensional vector. Thus $||X\hat{\alpha} - y||_2^2$ is the square of the 2-norm of the n dimensional vector $X\hat{\alpha} - y$. To be more specific, it's a scalar in $\mathbb{R}$.

(d)

Reginald is not necessarily correct. He may have an issue with overfitting the data. Although it looks like X predicts values for y well, Reginald hasn't tested X's prediction for a new value not in y. Cross-validation can be used here to ensure that X makes good predictions without overfitting.

Question 3

(a)

```
def func1(x, y):
  return (x - y)**2 + x*y

def func1_grad(vx, vy):
  dfdx = 2*vx - vy
  dfdy = 2*vy - vx
  return np.array([dfdx, dfdy])

########## f1 ##########

#initialize location and settings
v_init = np.array([2, 3])
num_iter = 20
values = np.zeros([num_iter, 2])

values[0,:] = v_init
v = v_init

gamma = 0.05

# actual gradient descent algorithm
for i in range(1, num_iter):
  v = v - gamma * func1_grad(v[0],v[1])
  values[i, :] = v
```

```
  print(func1(v[0], v[1]))

print('\n')
```

Function values for each step:
6.1825
5.482168749999999
4.877186453125
4.350745589335937
3.8897616734954097
3.48393186574361
3.125045871620225
2.8064799937505835
2.522824297718033
2.2696066633667744
2.043087476992105
1.8401059402222188
1.6579641941916723
1.4943392365774435
1.347215344339452
1.2148316957210588
1.095641320174279
0.9882785453901884
0.8915328656811666

(b)

```
def func2(x, y):
  return (1 - (y - 4))**2 + 35*((x + 6) - (y - 4)**2)**2

def func2_grad(vx, vy):
  dfdx = 70*(vx - (vy - 4)**2 + 6)
  dfdy = 2*((-70)*(vy - 4)*(vx - (vy - 4)**2 + 6) + vy - 5)
  return np.array([dfdx, dfdy])

########## f2 ##########

#initialize location and settings
v_init = np.array([0, 2])
num_iter = 100
values = np.zeros([num_iter, 2])

values[0, :] = v_init
v = v_init

gamma = 0.0015

# actual gradient descent algorithm
for i in range(1,num_iter):
  v = v - gamma * func2_grad(v[0], v[1])
  values[i,:] = v
  print(func2(v[0], v[1]))

print('\n')
```

Function values for each step:
187.88006849523507
507.0224399254285
97.4687922498154
237.49697288060918
163.67331237471922
397.5295607383085
92.91856638516737
203.84546587090867
124.44105580352209
271.080602278776
98.95907069068228
201.39078707217467
97.62861798226827
191.17834664797255
88.27719160868917
164.68270551157514
81.25126770491809
145.1001259745785
74.5714162236261
127.68726031521832
68.41053884855933
112.52686890847815
62.73791418295255
99.31273134192851
57.52379855993894
87.78384896359964
52.739191446705306
77.71665530217534
48.355989273410415
68.91925835864996
44.347125359484906
61.226658666804866
40.68667602849454
54.496735715292644
37.349934338528655
48.60686819081629
34.31345506468583
43.451081187215244
31.555074746514627
38.93763293113783
29.053910457193865
34.98696898146169
26.790340723278902
31.5299844010636
24.745971784329548
28.506544682212798
22.903592151697364
25.86422466532266
21.24711821379695
23.557231662398088
19.76153344310416
21.545484748545707
18.43282358438362

19.79382693000119
17.24791003621831
18.271350804783943
16.19458346727958
16.950821537976655
15.261439521275562
15.808183587525352
14.43781824628653
14.822139729090663
13.713748623429828
13.973792617426142
13.07989925935934
13.246340456271119
12.527535946853781
12.624819393264982
12.048486396374164
12.095886072745706
11.635112017061086
11.647634427091607
11.280286204802039
11.269441322940478
10.97737821016512
10.951836153071072
10.720241344280659
10.686389920354557
10.503204066801372
10.465619827931777
10.321062408896232
10.282905887018668
10.169072225061697
10.132416583721625
10.042939934603105
10.009041199220494
9.938810686491264
9.908326933963064
9.853253227611802
9.826419520702043
9.783241134806245
9.760006496960527
9.72613044490201
9.706262721998378
9.67963404785849
9.662798050615192
9.641793468778047
9.627607308926443
9.61094883880406

(c)

```
########## f1 ##########

#initialize location and settings
v_init = np.array([2, 3])
```

```
num_iter = 20
values = np.zeros([num_iter, 2])

values[0,:] = v_init
v = v_init

gamma = 0.2

# actual gradient descent algorithm
for i in range(1, num_iter):
  v = v - gamma * func1_grad(v[0],v[1])
  values[i, :] = v
  print(func1(v[0], v[1]))

print('\n')
```

Function value after 20 steps:
0.0012980742146342732

(d)

```
########## f2 ##########

#initialize location and settings
v_init = np.array([0, 2])
num_iter = 100
values = np.zeros([num_iter, 2])

values[0, :] = v_init
v = v_init

gamma = 0.002201

# actual gradient descent algorithm
for i in range(1, num_iter):
  v = v - gamma * func2_grad(v[0], v[1])
  values[i,:] = v
  print(func2(v[0], v[1]))

print('\n')
```

Function value after 100 steps:
0.5027520844564334