Homework 4

Kyle Kazemini

November 10, 2020

(1)

(a)

```
import pandas as pd
import numpy as np
from numpy import linalg as LA
from scipy import linalg
from google.colab import files

files.upload()
files.upload()
files.upload()

# Question 1

x = pd.read_csv("X4.csv")
y = pd.read_csv("y4.csv")

# part (a)

def func(x, y):
  return x + y

def func_grad(vx, vy):
  dfdx = vx
  dfdy = vy
  for i in range(1, len(x)):
    dfdx = dfdx + x.iloc[i][0]
    dfdy = dfdy + y.iloc[i][0]
  return np.array([dfdx, dfdy])

#initialize location and settings
v_init = np.array([29,4])
num_iter = 10
values = np.zeros([num_iter,2])

values[0,:] = v_init
v = v_init

gamma = 0.000001

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

```python
print(values)
```

[[29. 4.] [28.999943 3.99975129] [28.999886 3.99950258] [28.999829 3.99925387] [28.999772 3.99900516] [28.999715 3.99875645] [28.999658 3.99850775] [28.999601 3.99825904] [28.999544 3.99801033] [28.999487 3.99776162]]

(b)

```python
# part (b)

def func(x, y):
  return x + y

def func_grad(vx, vy):
  dfdx = vx
  dfdy = vy
  return np.array([dfdx, dfdy])

#initialize location and settings
v_init = np.array([5,4])
num_iter = 10
values = np.zeros([num_iter,2])

values[0,:] = v_init
v = v_init

gamma = 0.00025

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

print(values)
```

[[5. 4.] [4.99999875 3.999999] [4.9999975 3.999998] [4.99999625 3.999997] [4.999995 3.999996] [4.99999375 3.999995] [4.9999925 3.999994] [4.99999125 3.999993] [4.99999 3.999992] [4.99998875 3.999991]]

I prefer incremental gradient descent. I think it's a little bit easier to implement and to explain. Although the results of the data analysis method of choice are very important, I think ease of use/implementation and complexity should also be considered. It may not be worth it to get marginally better results if it's much harder to understand and explain to a non-expert. I'm thinking generally, but I prefer incremental gradient descent for these reasons.

(2)

(a)

False. The left singular vectors of A are in $\mathbb{R}^{100}$.

(b)

True. The right singular vectors of A are in $\mathbb{R}^8$ and this fact follows from the properties of singular vectors.

(d)

$rank(B) = 2$. From the textbook, we know that $u_j v_j^T$ is an $n\ x\ d$ (in this case 100 x 8) matrix with rank 1. Thus, B is the sum of two rank 1 matrices, so $rank(B) = 2$.

(e)

$B \in \mathbb{R}^{100x8}$. That is, B is a 100 x 8 matrix. It has 100 rows and 8 columns.

(f)

$\|Bv_3\| = 0$. We know that V is an orthogonal matrix, which means that the inner product of any two columns of V is 0. v_3 is one of these columns and the matrix B is constructed from two of the columns v_1 and v_2. Thus, the inner product of any column of B with any column of V is 0. This implies that $\|Bv_3\| = 0$.

(3)

(a)

```python
# Question 3

A = pd.read_csv("A.csv")

A = np.matrix(A)
U, s, VT = LA.svd(A)

# part (a)

print(s[2]) # The second singular value of A.
```

4.044276734606446

(b)

```python
# part (b)
print(LA.matrix_rank(A))
# Equivalently, the rank of A is the number of non-zero values in the diagonal of S, which, of
    course, is also 6.
```

6

(c)

```python
# part (c)

# This is one of the formulations for eigenvalues and eigenvector in relation to SVD given in the
    textbook.
V = VT.transpose()
ATA = A.transpose() @ A
ATAV = ATA @ V

values, vectors = LA.eig(ATAV)
values = np.array(values)
vectors = np.matrix(vectors)

for item in values: # iterates over the list of eigenvalues
  print(item)

for item in np.nditer(vectors): # iterates over the list of eigenvectors
  print(item)
```

Eigenvalues:

(5.059811832100812+74.07158537850441j)
(5.059811832100812-74.07158537850441j)
(-6.382863207749627+4.604796316400085j)
(-6.382863207749627-4.604796316400085j)
(0.11425707371163266+0.8888922016380596j)
(0.11425707371163266-0.8888922016380596j)
(-1.56983268123683e-14+0j)

Eigenvectors:

(0.04621761376141394-0.19643866141982957j)
(0.04621761376141394+0.19643866141982957j)
(-0.014647891841361293+0.009796886169846628j)
(-0.014647891841361293-0.009796886169846628j)
(0.0005323507399805442+0.00016061845788768769j)
(0.0005323507399805442-0.00016061845788768769j)
(-5.056306957211146e-19+0j)

(0.22143897303542198+0.2318447502598765j)
(0.22143897303542198-0.2318447502598765j)
(0.027643605206448714-0.008746534563436398j)
(0.027643605206448714+0.008746534563436398j)
(-0.0015853676849558835+0.001623538058213208j)
(-0.0015853676849558835-0.001623538058213208j)
(7.37569259419523e-17+0j)

(-0.05661834016860583-0.1886957470177858j)
(-0.05661834016860583+0.1886957470177858j)
(-0.024837401398638798-0.18443336857974846j)
(-0.024837401398638798+0.18443336857974846j)
(0.012085721261557627-0.02499082867527697j)

(0.012085721261557627+0.02499082867527697j)
(1.0813967801868786e-16+0j)

(0.7924700362053702+0j)
(0.7924700362053702-0j)
(-0.7885572562456546+0j)
(-0.7885572562456546-0j)
(0.0522941300381117+0.12641401716434617j)
(0.0522941300381117-0.12641401716434617j)
(1.1370940669304311e-15+0j)

(0.24125237669841076-0.006507102898729435j)
(0.24125237669841076+0.006507102898729435j)
(-0.18747780301039746-0.39441207306474035j)
(-0.18747780301039746+0.39441207306474035j)
(0.1107537917827409+0.40422242818391957j)
(0.1107537917827409-0.40422242818391957j)
(2.135996058496008e-15+0j)

(0.0034091889171940226-0.15631553672156234j)
(0.0034091889171940226+0.15631553672156234j)
(0.30109628118623755+0.020922524751886835j)
(0.30109628118623755-0.020922524751886835j)
(0.8489173061929061+0j)
(0.8489173061929061-0j)
(-3.970737886194188e-14+0j)

(-0.23033341549060007+0.23221411299573305j)
(-0.23033341549060007-0.23221411299573305j)
(0.2297712189788535+0.08827985908846284j)
(0.2297712189788535-0.08827985908846284j)
(0.01893095623975588-0.28952997753322124j)
(0.01893095623975588+0.28952997753322124j)
(1+0j)

(d)

```
# part (d)

# Calculate the Frobenius norm using the definition from the textbook.
sum = 0

for i in s:
  if i > 4:
    sum = sum + i**2

sum
```

$\|A - A_k\|_F^2 = 448.56785656660077$.

(e)

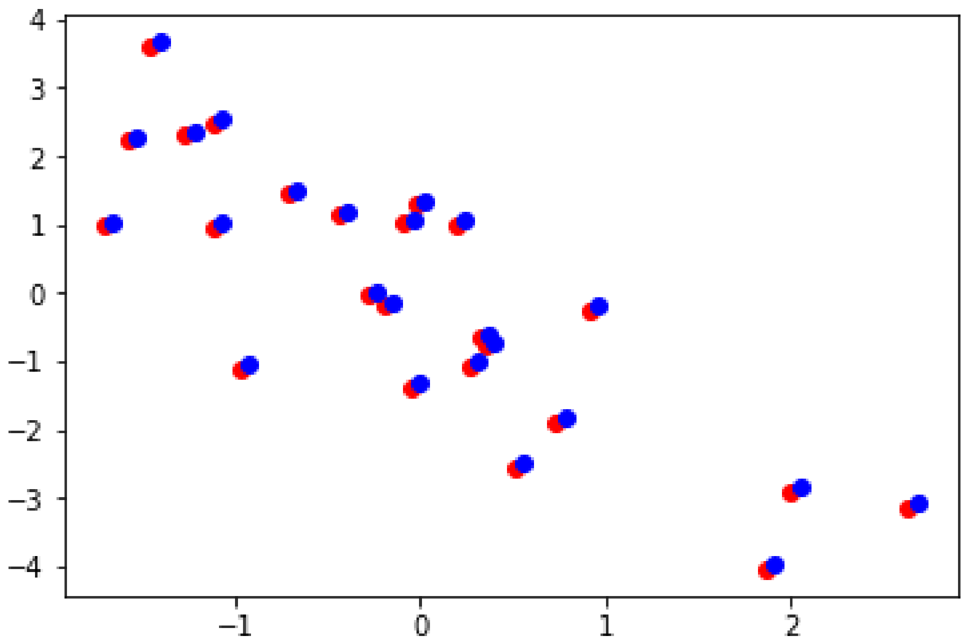
```
# part (e)

S = linalg.diagsvd(s, 7, 7)

# Calculate the matrices for A_k where k = 3
UK = np.delete(U, np.s_[3:24], axis=1)
print(UK.shape)

S = np.delete(S, np.s_[3:7], axis=0)
SK = np.delete(S, np.s_[3:7], axis=1)
print(SK.shape)

VK = np.delete(V, np.s_[3:7], axis=1)
print(VK.shape)

AK = UK @ SK @ VK.transpose()

print(LA.norm(A - AK))
```

(24, 3)
(3, 3)
(7, 3)

$\|A - A_k\|_2^2 = 2.247342094297741.$

(f)

```
# part (f)

c = A.mean(0)

C = np.eye(24) - np.ones(24)/24
Acc = C @ A

import matplotlib.pyplot as plt
plt.plot(A[:,0],A[:,1],'ro')
plt.plot(Acc[:,0],Acc[:,1],'bo')
plt.show()

U, s, VT = LA.svd(Acc)
V = VT.transpose()

# Best 3-dimensional subspace
B = np.delete(V, np.s_[3:7], axis=1)
print(B)

# Calculate the Frobenius norm using the definition from the textbook.
sum = 0

for i in s:
  if i > 4:
    sum = sum + i**2

sum
```



[[0.17550399 0.38744075 0.05760977] [-0.4910176 -0.34449428 -0.35297731] [0.29874973 0.32218147 -0.19733465] [-0.76211995 0.49505772 0.22849516] [-0.16960735 0.12623284 -0.74556077] [0.16130215 0.20380844 -0.4743991] [-0.05689294 -0.56996289 0.00466516]]

$\|A - \pi_B(A)\|_F^2 = 425.3664848278037.$

(g)

```
# part (g)

print(LA.norm(A - Acc))
```

$\|A - \pi_B(A)\|_2^2 = 3.2989634685980516.$