COM SCI 260B HW 3 Solution

Ashish Kumar Singh (UID:105479019)

May 10, 2022

Problem 1. Show that $\max_{v:||v||=1} ||Xv|| \leq \sigma_1$

Solution 1. Given $X = U\Sigma V^T$, we can write,

$$X = \sum_i \sigma_i u_i v_i^T \tag{1}$$

We can write vector v in the basis of columns of V, i.e.

$$v = \sum_i \alpha_i v_i \tag{2}$$

$$||v||^2 = \sum_i \alpha_i^2 ||v_i||^2$$

Since, v_i are orthonormal and $||v|| = 1$,

$$||v||^2 = \sum_i \alpha_i^2 = 1 \tag{3}$$

Combining eqn (1) and (2),

$$Xv = \sum_i \sigma_i u_i v_i^T . \sum_i \alpha_i v_i$$

Since v_i and u_i are orthonormal,

$$Xv = \sum_i \sigma_i \alpha_i u_i v_i^T v_i$$

$$Xv = \sum_i \sigma_i \alpha_i u_i$$

$$||Xv||^2 = \sum_i \sigma_i^2 \alpha_i^2 ||u_i||^2$$

$$||Xv||^2 = \sum_i \sigma_i^2 \alpha_i^2$$

By definition, σ_1 is greatest of all the singular values, hence $\sigma_i \leq \sigma_1$

$$||Xv||^2 \leq \sigma_1^2 \sum_i \alpha_i^2$$

$$||Xv||^2 \leq \sigma_1^2$$

$$||Xv|| \leq \sigma_1$$

Problem 2. Best-fit subspace dimension k

Solution 2. Lets assume that first $k-1$ singular vectors gives a best-fit subspace of dimension $k-1$, which means if $v_1, v_2, ..., v_{k-1}$ are the first $k-1$ singular vectors then for any vectors $w_1, w_2, ..., w_{k-1}$, we have:

$$\sum_{i=1}^{k-1} ||Xw_i||^2 \leq \sum_{i=1}^{k-1} ||Xv_i||^2 \tag{4}$$

Now, lets say that S^* is the best fit subspace of dimension k, we can choose the orthonormal basis for S^* such that $w_k \perp v_i$ for $i \in \{1, k-1\}$

$$S^* = Span(w_1, w_2, ..., w_k)$$

Lets say that v_k is the kth singular vector, then by definition

$$v_k = argmax_{||v||=1, v \perp v_i, i \in \{1,k-1\}} ||Xv||$$
$$||Xv_k|| \geq ||Xw_k|| \tag{5}$$

Lets say S is the span of first k singular vectors,

$$S = Span(v_1, v_2, ..., v_k)$$

On combining eqn 4 and 5 we get,

$$\sum_{i=1}^{k-1} ||Xw_i||^2 + ||Xw_k||^2 \leq \sum_{i=1}^{k-1} ||Xv_i||^2 + ||Xv_k||^2$$

$$\sum_{i=1}^{k} ||Xw_i||^2 \leq \sum_{i=1}^{k} ||Xv_i||^2$$
$$Var(S^*; X) \leq Var(S; X)$$

which shows that S maximizes var in dimension k, hence the span of first k right singular vectors gives the best-fit subspace of dimension k. We know that for $k = 2$ it is true, thus by induction it is true for every k.

Problem 3. Smallest Singular Vector

Solution 3. We first find the largest singular vector and corresponding largest singular value σ_1 of X using Power Iteration method.
Now, we construct new matrix $Y = X^t X$, If $X = U\Sigma V^T$, then $Y = V\Sigma^2 V^T$, we can see that $YV = V\Sigma^2$ which means eigenvalues of Y are square of the singular values of X.
Since Y is symmetric, we can shift its singular value by shifting the matrix by scaled identity matrix.
We can form another matrix $Z = Y - \sigma_1^2 I$. Thus making the magnitude of smallest singular value the highest. Thus, on running Power Iteration method on Z, we will get the smallest right singular vector of X.

Problem 4. Singular Value Projection

Solution 4a.

$$L = \sum_{(i,j) \in O} (X_{ij} - Y_{ij})^2$$

$$\frac{\partial L}{\partial Y_{ij}} = \begin{cases} 0 & \text{for } i, j \notin O \\ 2(Y_{ij} - X_{ij}) & \text{for } i, j \in O \end{cases}$$

$$\frac{\partial L}{\partial Y_{ij}} = 2(Y_{ij} - X_{ij}).O_{ij}$$

$$\frac{\partial L}{\partial Y} = 2(Y - X).O$$

Solution 4b. code and plots attached at the end

Problem 5. Singular Value Projection

Solution 5. code and plots attached at the end

Mode	Number of Iteration	Time
Scipy SVD	-	16.17
PI	10	0.52
PI	20	1.07
PI	30	1.59
PI	40	2.13
PI	50	2.67
PI	60	3.20
PI	70	3.72
PI	80	4.25
PI	90	4.77
PI	100	5.30

Table 1: Time comparison for Scipy vs PI

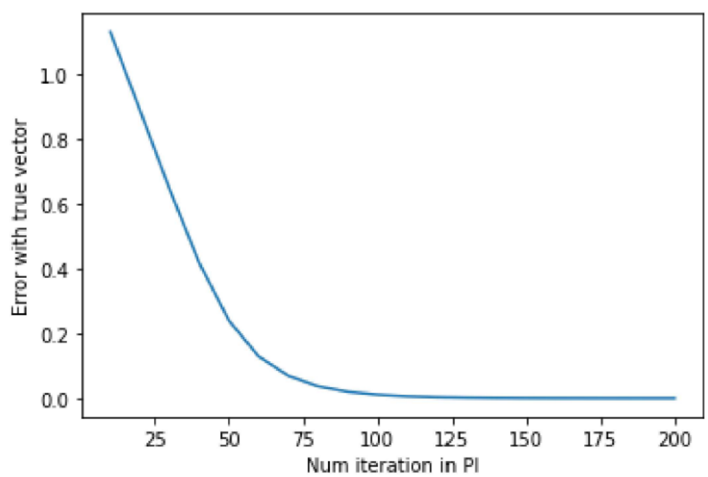


Figure 1: Plot of Error vs number of iteration in PI

In [2]:

```
import numpy as np
import scipy as scipy
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
from mpl_toolkits.mplot3d import Axes3D
import gzip
from sklearn.preprocessing import OneHotEncoder
from scipy.special import expit
import celluloid
from celluloid import Camera
from matplotlib import animation
from IPython.display import HTML
from matplotlib.lines import Line2D

np.random.seed(2022)
```

In [8]:

```
def lplot(Ys,labels=['1','2','3','4','5','6'],ylabel='Function value'):
    """Line plot of the Y values. (Same as above, but no animation).
    Ys is a list where each element is an array of numbers to plot.
    """
    colors = ['blue','red','green','black','cyan','purple','pink']
    fig, ax = plt.subplots(figsize=(6,6))
    T = len(Ys[0])
    #plt.yscale('log')
    handles = []
    for i in range(len(Ys)):
        handles.append(Line2D([0], [0], color=colors[i], label=labels[i]))
    plt.legend(handles = handles, loc = 'upper right')
    plt.xlabel('Step')
    plt.ylabel(ylabel)
    for j in range(len(Ys)):
        plt.plot(range(T),Ys[j][:T],color=colors[j],marker='o')
```

In [5]:

```
def gen_rank_k_matrix(n,d,k):
    U = np.random.normal(0,1,(n,k))
    V = np.random.normal(0,1,(k,d))
    X = U.dot(V)
    if np.linalg.matrix_rank(X) == k:
        return X
    else:
        return gen_rank_k_matrix(n,d,k)

def gen_mask(n,d,p):
    R = np.random.rand(n,d)
    O = np.zeros((n,d))
    O[R < p] = 1
    return O

def cost(X,Y,O):
    return np.sum((X - Y)**2)

def gradient_fn(X,Y,O):
    return 2*(Y*O - X)

def gradient_descent(xinit,steps,gradient):
    """Run gradient descent.
    Return an array with the rows as the iterates.
    """
    xs = [xinit]
    x = xinit
    for step in steps:
        x = x - step*gradient(x)
        u, s, vT = scipy.sparse.linalg.svds(x, k=5)
        x = u.dot(np.diag(s).dot(vT))
```

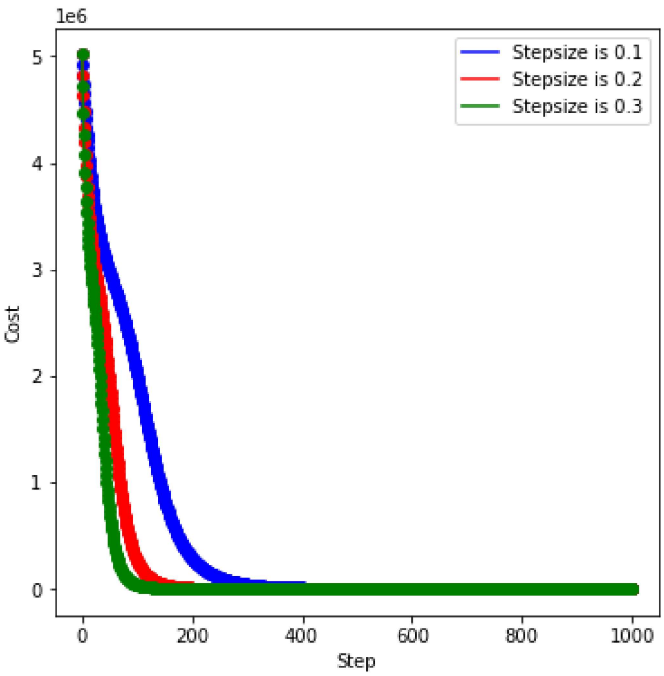
```
        xs.append(x)
    return np.array(xs)
```

In [6]:

```
n = 1000
d = 500
k = 5
p = 0.1
num_iter = 1000
X = gen_rank_k_matrix(n,d,k)
X_init = gen_rank_k_matrix(n,d,k)
O = gen_mask(n,d,p)
X_in = X*O
objective = lambda Y: cost(X, Y, O)
gradient = lambda Y: gradient_fn(X_in, Y, O)
step_sizes = [0.1,0.2,0.3]
labels = ['Stepsize is '+str(step) for step in step_sizes]
Xs = [gradient_descent(X_init,[size]*num_iter,gradient) for size in step_sizes]
Ys = [[objective(y) for y in x] for x in Xs]
```

In [9]:

```
lplot(Ys,labels,'Cost')
```



In [ ]:

In [1]:
```
import numpy as np
import sys
import scipy
from sklearn.preprocessing import OneHotEncoder
from scipy.special import expit
import time
import matplotlib.pyplot as plt
```

In [2]:
```
def gen_mask(n,d,p):
    R = np.random.rand(n,d)
    O = np.zeros((n,d))
    O[R < p] = 1
    return O
```

In [3]:
```
n = 10000
p = 0.01
k = 20
```

In [4]:
```
G = np.random.normal(0,1,(n,n))
O = gen_mask(n,n,p)
G = G*O
U = np.random.normal(0,1,(n,k))
```

In [5]:
```
Z = U.dot(U.T)+G
```

In [6]:
```
def power_iteration(U,G,v0,T):
    v = v0
    Ut = U.T
    Gt = G.T
    for i in range(T):
        u = U.dot(Ut.dot(v)) + G.dot(v)
        u = U.dot(Ut.dot(u)) + Gt.dot(u)
        v = u/np.linalg.norm(u)
    return v

def scipy_default(X):
    u,s,vt = scipy.sparse.linalg.svds(X,k=1)
    return vt.T
```

In [7]:
```
def run_scipy(X):
    start = time.time()
    v = scipy_default(X)
    end = time.time()
    t = end - start
    return v,t

def run_pi(U,G,v0,T):
    #v0 = np.random.normal(0,1,(G.shape[1],1))
    #v0 = v0/np.linalg.norm(v0)
    start = time.time()
    v = power_iteration(U,G,v0,T)
    end = time.time()
    t = end - start
    return v,t
```

In [8]:
```
t_scipy = 0.0
t_pi = [0.0]*20
v_pi = [0.0]*20
num_runs = 10
for i in range(num_runs):
```

```
        sys.stdout.write("%d / %d \r" %(i+1,num_runs))
        sys.stdout.flush()
        v,t = run_scipy(Z)
        t_scipy += t
        v0 = np.random.normal(0,1,(G.shape[1],1))
        v0 = v0/np.linalg.norm(v0)
        for j in range(20):
            v0,t = run_pi(U,G,v0,10)
            t_pi[j] += t
            v_pi[j] += min(np.linalg.norm(v0-v),np.linalg.norm(v0+v))
t_pi = [x/num_runs for x in t_pi]
v_pi = [x/num_runs for x in v_pi]
for j in range(1,20):
    t_pi[j] += t_pi[j-1]
print("Scipy svd time: ",t_scipy)
print("Power iteration time:",t_pi)
```

```
Scipy svd time:  16.172028303146362
Power iteration time: [0.5271583557128906, 1.0714661121368407, 1.59823579788208, 2.131119060516
3572, 2.6719422340393066, 3.200266790390015, 3.7280841112136844, 4.254444932937623, 4.778913688
659668, 5.303987979888916, 5.830049324035644, 6.355394983291625, 6.882603192329406, 7.405512714
385986, 7.932202959060668, 8.45703580379486, 8.983737850189208, 9.513005614280699, 10.039047312
736509, 10.566315460205075]
```

In [9]:

```
plt.xlabel('Num iteration in PI')
plt.ylabel('Error with true vector')
plt.plot(range(10,201,10),v_pi)
```

Out[9]: [<matplotlib.lines.Line2D at 0x1bfbda29130>]