CS 260B-80 | Assignment #3 | Jason Chapman

(1) To show $\max_{v:\|v\|=1} \|Xv\| \leq \sigma_1$ use SVD

$X = U\Sigma V^T$ // $\Sigma$ is $r \times r$ diagonal w/ $\Sigma_{11} \geq \Sigma_{22} \geq \ldots \geq \Sigma_{rr}$

& $\sigma_i = \Sigma_{ii}$

$Xv = (U\Sigma V^T)v$

∵ $U$ & $V$ are orthonormal, so let $v = Vw$ where $w$ has the same dimension as the number of columns of $V$

$\Rightarrow Xv = (U\Sigma V^T)Vw$

$= U\Sigma w$

$\Rightarrow \|Xv\| = \|U\Sigma w\|$

$\leq \|U\Sigma\| \cdot \|w\|$ // submultiplicative property $\|AB\| \leq \|A\| \cdot \|B\|$

$= \|U\Sigma\|$ // $\|w\| = \|V^T v\| = \|v\| = 1$

$\leq \|U\| \cdot \|\Sigma\|$

$= \|\Sigma\|$ // $U$ is orthonormal, $\|U\| = 1$

$= \sigma_1$ // ∵ $\|\Sigma\| = \max(\sigma_i)$ = largest singular value

$\Rightarrow \|Xv\| \leq \sigma_1$ ✓

(2) From lecture: Span of first 2 right singular vectors gives a best-fit subspace in 2 dimensions

- Assume span of first k-1 right singular vectors gives a best-fit subspace in k-1 dimensions

Prove for k:

Let $S^*$ be optimal subspace with k dimensions

To find best-fit subspace need to find k orthonormal vectors that span $S^*$

Let $w_1, w_2, \dots, w_k$ be an orthonormal basis for $S^*$ such that each $w_k$ $(k>2)$ is $\perp$ to all other vectors $w_1, \dots w_{k-1}$

// We can do this similar to how we chose $w_2$ to be $\perp$ to $v_1$ for the $k=2$ case

By choosing this basis, we have ensured that any vector in $S^*$ can be a linear combination of $w_1, \dots, w_k$, which means that the span of the first k right singular vectors gives a best-fit subspace for a problem in k dimensions

✓

(3)

$Y = X^T X \quad , \quad X = U \Sigma V^T$

$$\Rightarrow Y = (U\Sigma V^T)^T (U \Sigma V^T)$$

$$= (V \Sigma U^T U \Sigma V^T) = V \Sigma^2 V^T$$

// SVs of $Y$ are $\sigma_1^2, \sigma_2^2, \ldots \sigma_n^2$

- Subtract the smallest eigenvalue from all of the eigenvalues of $Y$. This makes the smallest eigenvalue zero, which corresponds to the smallest singular value of $X$.

  After this shift, we compute the eigenvectors of the modified matrix $Y$. The eigenvector corresponding to the eigenvalue of zero will be the smallest right singular vector of $X$.

④ a) $L(Y) = \sum_{(i,j)\in O} (X_{ij} - Y_{ij})^2$

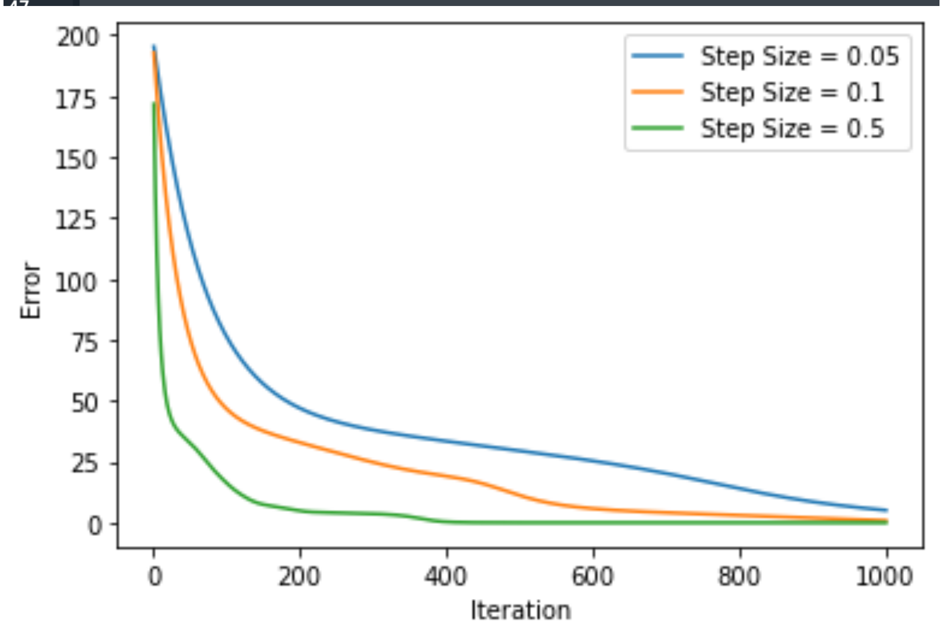
$\nabla L(Y) \in \mathbb{R}^{n\times d}$

$\nabla L(Y)_{ij} = 2(X_{ij} - Y_{ij})\cdot(-1)$

$\Rightarrow \quad \nabla L(Y)_{ij} = 2(Y_{ij} - X_{ij})$

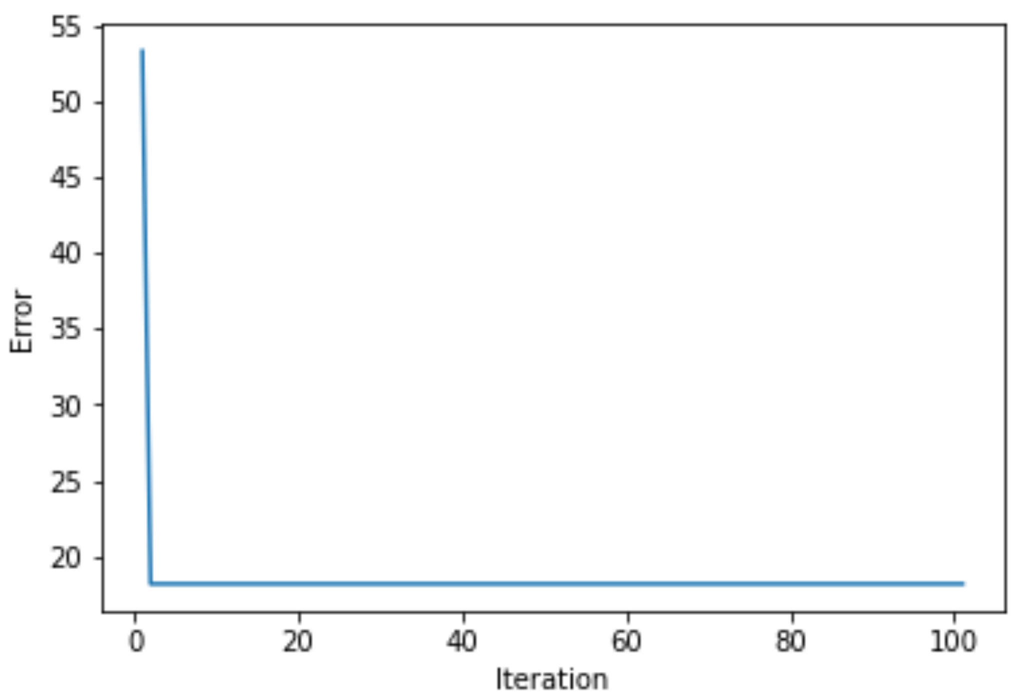
4b.)

```
import numpy as np
import scipy
import matplotlib.pyplot as plt

k = 5
n = 1000
d = 500
p = 0.1
iterations = 1000

U = np.random.rand(n, k)
V = np.random.rand(k, d)
X = np.dot(U, V)

def singular_value_projection(X, O, eta, T):
    n, d = X.shape
    Y = np.random.rand(n, d)
    error_list = []

    for i in range(T):
        gradient = np.multiply(O, 2*(Y - X))
        Y -= eta*gradient

        U, S, VT = scipy.sparse.linalg.svds(Y, k = 5)
        Y = U @ np.diag(S) @ VT

        error = np.linalg.norm(np.multiply(O, Y - X))
        error_list.append(error)

    return Y, error_list


O = np.random.choice([0, 1], size=(n, d), p=[1 - p, p])
Y_hat1, err1 = singular_value_projection(X, O, .05, iterations)
Y_hat2, err2 = singular_value_projection(X, O, .1, iterations)
Y_hat3, err3 = singular_value_projection(X, O, .5, iterations)


plt.plot(range(1, iterations + 1), err1, label="Step Size = 0.05")
plt.plot(range(1, iterations + 1), err2, label="Step Size = 0.1")
plt.plot(range(1, iterations + 1), err3, label="Step Size = 0.5")
plt.xlabel("Iteration")
plt.ylabel("Error")
plt.legend()
plt.show()
```



5.)

| Method | Time to Run (s) |
|---|---|
| Built-In SVD | 1.746903896331787 |
| Power Iteration – 10 iterations | 0.03452355861663818 |
| Power Iteration – 20 iterations | 0.06516032218933106 |
| Power Iteration – 30 iterations | 0.10776979923248291 |
| Power Iteration – 40 iterations | 0.12278242111206054 |
| Power Iteration – 50 iterations | 0.19508419036865235 |
| Power Iteration – 60 iterations | 0.17753887176513672 |
| Power Iteration – 70 iterations | 0.21242260932922363 |
| Power Iteration – 80 iterations | 0.2983795881271362 |
| Power Iteration – 90 iterations | 0.3307294130325317 |
| Power Iteration – 100 iterations | 0.4645735263824463 |




This result is not satisfactory for me. However, I am not sure what I am doing wrong. My last vector at the completion of power iteration is very close to the true vector obtained from the built-in SVD, however I still get this constant error around 18. It seems that my power iteration algorithm converges in about 3 steps, but there is still the error given the small discrepancies in the two large dimension vectors. I am confident that I am doing everything correctly but am still not producing results that make sense to me. I have attached my code.

```python
import numpy as np
import scipy
import scipy.sparse as sp_sparse
import time
import matplotlib.pyplot as plt

def power_iteration(U,G,v0,T=100):
    vs = [v0]
    v = v0
    Ut = np.transpose(U)
    for i in range(T):
        u = Ut.dot(v)
        u = U.dot(u)
        u = u + G.dot(v)
        v = u/np.linalg.norm(u)
        vs.append(v)
    return np.array(vs)


n = 10000
k = 20
p = 0.01

G = sp_sparse.random(n, n, density=p, format='coo')
U = np.random.rand(n, k)
Z = np.dot(U, U.T) + G

start_time = time.time()
utrue,strue,vtrue = scipy.sparse.linalg.svds(Z,k=1)
end_time = time.time()
builtin_time = end_time - start_time

num_iterations = np.array([10, 20, 30, 40, 50, 50, 60, 70, 80, 90, 100])
time_list = []
error_list = []
for iterations in num_iterations:
    start_time = time.time()
    for j in range(10):
        v0 = np.random.rand(n,1)
        v0 = v0/np.linalg.norm(v0)
        Vs = power_iteration(U,G,v0,iterations)
    end_time = time.time()
    execution_time = (end_time - start_time)/10
    time_list.append(execution_time)
error = [np.linalg.norm(abs(v) - abs(vtrue)) for v in Vs]
```