Homework 7

7.1. Consider Example 7.17, p.318. Compute vectors of cosines, for each subspace approximations, i.e., with A_k where $k = 1, 2, \cdots, 5$.

Example 7.17. Recall Example 7.1. Consider the term-document matrix $A \in \mathbb{R}^{10\times 5}$ and the query vector $\mathbf{q} \in \mathbb{R}^{10}$, of which the query is "**ranking** of **web pages**". See pages 302–303 for details.

Document 1:	The **Google matrix** P is a model of the **Internet**.
Document 2:	P_{ij} is nonzero if there is a **link** from **web page** j to i.
Document 3:	The **Google matrix** is used to **rank** all **web pages**.
Document 4:	The **ranking** is done by solving a **matrix eigenvalue** problem.
Document 5:	**England** dropped out of the top 10 in the **FIFA ranking**.

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{10\times 5}, \qquad \mathbf{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \in \mathbb{R}^{10}.$$

Results:

cosine vector for subspace approximation A_1:
[1. 1. 1. 1. 1.]

cosine vector for subspace approximation A_2:
[0.78567263 0.83318895 0.96701013 0.48728872 0.18188745]

cosine vector for subspace approximation A_3:
[0.10236965 0.85005675 0.83708519 0.42181788 0.468531]

cosine vector for subspace approximation A_4:
[0.05399144 0.84759387 0.82907251 0.41654825 0.39175232]

cosine vector for subspace approximation A_5:
[-2.40579263e-16 7.22315119e-01 8.39254327e-01 3.61157559e-01
  3.61157559e-01]

HW7-1.py

```python
import numpy as np
```

```python
# really need to figure out the syntax for einsum to avoid this
def col_by_row_mult(u,v):
    n = np.size(u)
    if np.size(v) == n and u.ndim == 1 and v.ndim == 1:
        A = np.zeros((n,n))
        for i in range(n):
            A[i,:] = u[i]*v
        return A
    raise Exception('Arguments must be one dimensional vectors of the same size!')

def get_low_rank_svd(U, V, Sigma, rank):
    m, n = U.shape[0], V.shape[1]
    Ak = np.zeros((m, n))
    for i in range(rank):
        Ak += Sigma[i]*col_by_row_mult(U[:,i], V[:,i])
    return Ak

def get_document_matrix(Sigma, V_T, rank):
    return np.diag(Sigma[:rank]) @ V_T[:rank]

def get_query(U, q, rank):
    return U[:,:rank].T @ q

def get_cosines(U, Sigma, V_T, q, rank):
    identity = np.eye(V_T.shape[1]) # nxn identity matrix
    Dk = get_document_matrix(Sigma, V_T, rank)
    qk = get_query(U, q, rank)
    cosines = np.zeros(Dk.shape[1])
    for j in range(len(cosines)):
        temp = Dk @ identity[:,j]
        cosines[j] = np.dot(qk, temp)/(np.linalg.norm(qk)*np.linalg.norm(temp))
    return cosines


if __name__ == '__main__':
    A = np.array([[0,0,0,1,0],
                  [0,0,0,0,1],
                  [0,0,0,0,1],
                  [1,0,1,0,0],
                  [1,0,0,0,0],
                  [0,1,0,0,0],
                  [1,0,1,1,0],
                  [0,1,1,0,0],
                  [0,0,1,1,1],
                  [0,1,1,0,0]])
```

```python
q = np.array([0,0,0,0,0,0,0,1,1,1])
U, Sigma, V_T = np.linalg.svd(A, full_matrices=False)
'''print(f'shape of U: {U.shape}')
print(f'shape of Sigma: {Sigma.shape}')
print(f'shape of V.T: {V_T.shape}')'''
'''SVD_test = U @ np.diag(Sigma) @ V_T
SVD_test[np.abs(SVD_test) < 10e-12] = 0
print(SVD_test)'''

rank = A.shape[1]
m, n = A.shape
cosines = np.zeros((rank,n))
for k in range(rank):
    cosines[k] = get_cosines(U, Sigma, V_T, q, k+1)
    print(f'cosine vector for subspace approximation A_{k+1}: \n{cosines[k]}\n')
```

7.2. Verify equations in Derivation 7.30, p.327, particularly (7.38), (7.39), and (7.40).

$\boxed{\textbf{Derivation}}$ **7.30.** Let $\mathbf{z} = G\mathbf{y}$.

- **Normalization-free**: Since G is column-stochastic ($\mathbf{e}^T G = \mathbf{e}^T$),

$$||\mathbf{z}||_1 = \mathbf{e}^T\mathbf{z} = \mathbf{e}^T G\mathbf{y} = \mathbf{e}^T\mathbf{y} = ||\mathbf{y}||_1. \tag{7.37}$$

  Thus, when the power method begins with $\mathbf{y}^{(0)}$ with $||\mathbf{y}^{(0)}||_1 = 1$, the normalization step in the power method is unnecessary.

- Let us look at the multiplication in some detail:

$$\mathbf{z} = \left[\alpha P + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T\right]\mathbf{y} = \alpha Q\mathbf{y} + \beta\frac{\mathbf{e}}{n}, \tag{7.38}$$

  where

$$\beta = \alpha\mathbf{d}^T\mathbf{y} + (1-\alpha)\mathbf{e}^T\mathbf{y}. \tag{7.39}$$

- Apparently we need to know which pages lack outlinks ($\mathbf{d}$), in order to find β. However, in reality, we do not need to define $\mathbf{d}$. It follows from (7.37) and (7.38) that

$$\beta = 1 - \alpha\mathbf{e}^T Q\mathbf{y} = 1 - ||\alpha Q\mathbf{y}||_1. \tag{7.40}$$

Consider $z = Gy$ where G is the Google matrix and y is the Pagerank vector.

$$z = Gy \iff z = \left[\alpha P + (1-\alpha)\frac{1}{n}ee^T\right]y$$
$$z = \alpha Py + (1-\alpha)\frac{1}{n}ee^Ty$$

By definition, the column-stochastic matrix $P = Q + \frac{1}{n}ed^T$.

$$z = \alpha\left[Q + \frac{1}{n}ed^T\right]y + (1-\alpha)\frac{1}{n}ee^Ty$$
$$z = \alpha Qy + \alpha\frac{1}{n}ed^Ty + (1-\alpha)\frac{1}{n}ee^Ty$$

Let $v = \frac{e}{n}$ be a personalization vector s.t. $\|v\|_1 = 1$ for convenience.

$$z = \alpha Qy + \alpha vd^Ty + (1-\alpha)ve^Ty$$
$$z = \alpha Qy + v(\alpha d^Ty + (1-\alpha)e^Ty)$$

Let $\beta = \alpha d^Ty + (1-\alpha)e^Ty$. Then the above becomes

$$z = \alpha Qy + \frac{e}{n}\beta$$

This derivation consists of both 7.38 and 7.39.

For 7.40, consider the definition of β above

$$\beta = \alpha d^Ty + (1-\alpha)e^Ty \implies \alpha d^Ty + e^Ty - \alpha e^Ty$$
$$= -\alpha(e^Ty - d^Ty) + e^Ty = -\alpha(e^T - d^T)y + e^Ty$$

Since e^TQ is essentially the binary complement of d^T, which can be represented as $\sim d^T = e^T - d^T$, $e^TQ = e^T - d^T$. So the above becomes

$-\alpha(e^T - d^T)y + e^Ty = -\alpha e^TQy + e^Ty$. Also note that $e^Ty = \|y\|_1 = 1$.

So $\beta = 1 - \alpha e^TQy$, verifying 7.40.

7.3. Consider the link matrix Q in (7.4) and its corresponding link graph in Figure 7.2. Find the pagerank vector $\mathbf{r}$ by solving the Google pagerank equation.

- You may initialize the power method with any vector $\mathbf{r}^{(0)}$ satisfying $||\mathbf{r}^{(0)}||_1 = 1$.
- Set $\alpha = 0.85$.
- Let the iteration stop, when `residual` $< 10^{-4}$.

- The following **link graph** illustrates a set of web pages with outlinks and inlinks.

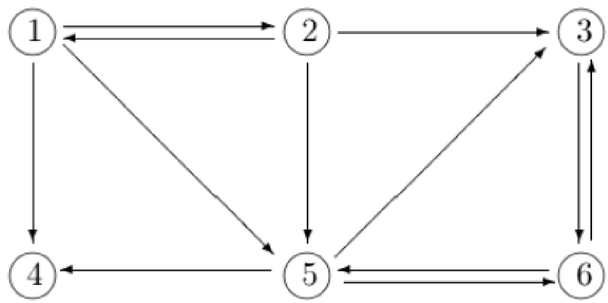

Figure 7.2: A link graph, for six web pages.

The corresponding **link matrix** becomes

$$Q = \begin{bmatrix} 0 & 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 1/3 & 1/2 \\ 1/3 & 0 & 0 & 0 & 1/3 & 0 \\ 1/3 & 1/3 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 1 & 0 & 1/3 & 0 \end{bmatrix} \tag{7.4}$$

Code and results given below for both 7.3 and 7.4, where each are handled in main().

Note: because of the tolerance required by the problems, results could not be as accurate as preferred. This is accounted for in tests by letting the parameter *rtol* in *np.allclose* be equal to the tolerance defined in these problems.

Also, tests are included for the irreducibility of P, but due to an issue with *networkx* in creating graphs for matrices such as P, it has been commented out. Since it’s not a hard requirement and the algorithm converges, I haven’t fully implemented it due to a lack of time.

7.4. Now, consider a **modified** link matrix $\widetilde{Q}$, by adding an outlink from page ④ to ⑤ in Figure 7.2. Find the pagerank vector $\widetilde{\mathrm{r}}$, by setting parameters and initialization the same way as for the previous problem.

- Compare r with $\widetilde{\mathrm{r}}$.
- Compare the number of iterations for convergence.

- The following **link graph** illustrates a set of web pages with outlinks and inlinks.

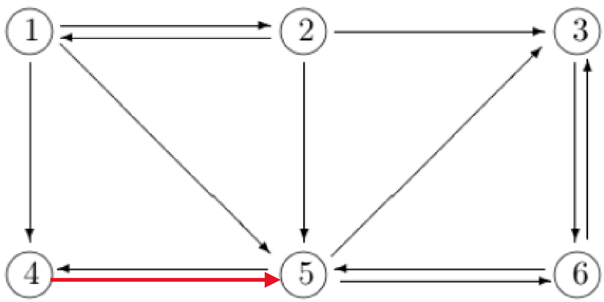

Figure 7.2: A link graph, for six web pages.

The corresponding **link matrix** becomes

$$Q = \begin{bmatrix} 0 & 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 1/3 & 1/2 \\ 1/3 & 0 & 0 & 0 & 1/3 & 0 \\ 1/3 & 1/3 & 0 & 1 & 0 & 1/2 \\ 0 & 0 & 1 & 0 & 1/3 & 0 \end{bmatrix} \tag{7.4}$$

Results:

Problem 7.3:
power method converged in 12 iterations
Pagerank vector: r =
[0.05791362 0.05791368 0.24899261 0.1165508 0.20676924 0.31186004]
Google matrix times Pagerank vector: Gr =
[0.05792024 0.05792022 0.24904538 0.11650484 0.20686962 0.3117397]
Gr = r? -> True

Problem 7.4:
power method converged in 12 iterations
Pagerank vector: r =
[0.03488372 0.03488378 0.24196065 0.1114054 0.26989359 0.30697286]
Google matrix times Pagerank vector: Gr =
[0.03488374 0.03488372 0.24181705 0.11135357 0.26992551 0.30713641]

Gr = r?   ->   True

Code:

HW7-3.py

```
import numpy as np

import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from copy import deepcopy

class IDD_test(object):
    def __init__(self, A):
        self.A = A
        self.A_shape = A.shape

    def make_dir_graph(self):
        rows, cols = np.where(self.A != 0)
        edges = zip(list(rows), list(cols))
        self.graph = nx.DiGraph()
        self.graph.add_edges_from(edges)

    def draw_graph(self):
        if not hasattr(self, 'graph'):
            self.make_dir_graph()
        node_labels = dict(enumerate([str(i+1) for i in range(self.A_shape[0])]))
        nx.draw(self.graph, node_size=500, labels = node_labels)
        plt.show()

    def is_irreducible(self):
        self.make_dir_graph()
        return nx.is_strongly_connected(self.graph)

    def is_diag_dominant(self):
        m = self.A.shape[0]
        Lambda = np.zeros(m)
        dominance_test = np.zeros(m, dtype=bool)
        strictness_test = np.zeros(m, dtype=bool)
        for i in range(m):
            Lambda[i] = np.linalg.norm(self.A[i], ord=0) - np.abs(self.A[i,i])
            dominance_test[i] = (np.abs(self.A[i,i]) >= Lambda[i])
            strictness_test[i] = (np.abs(self.A[i,i]) > Lambda[i])
        return (sum(dominance_test) == m) and (sum(strictness_test) > 0)

    def is_IDD(self):
```

```
        return self.is_irreducible() and self.is_diag_dominant()

# really need to figure out the syntax for einsum to avoid this
def col_by_row_mult(u,v):
    n = np.size(u)
    if np.size(v) == n and u.ndim == 1 and v.ndim == 1:
        A = np.zeros((n,n))
        for i in range(n):
            A[i,:] = u[i]*v
        return A
    raise Exception('Arguments must be one dimensional vectors of the same size!')

def get_P_matrix(Q, n):
    e = np.ones(n)
    d = np.invert(np.array([sum(Q[:,j]) for j in range(n)], dtype=bool)).astype(int) #
possibly incorrect
    # column-stochastic matrix of which columns are probability vectors
    P = Q + (1/n)*(col_by_row_mult(e, d))
    # test below fails for modified Q - need to figure out how to fix it or delete the
class above
        # might just be because P = Q in that case
    '''P_tester = IDD_test(P)
    if not P_tester.is_irreducible():
        raise Exception(f'ERROR: stochastic matrix P formed from Q must be
irreducible!')'''
    return P


def get_Google_matrix(P, alpha, n):
    # must be irreducible if P is irreducible
    return alpha*P + ((1-alpha)/n)*np.ones((n,n))

def power_method(Q, alpha, r_init, n, TOL):
    r = deepcopy(r_init)
    e = np.ones(n)
    # personalization vector
    v = e/n
    residual = 1
    loop_count = 0
    # might switch to alpha**k >= TOL because residuals as high as 10e-4 gets awful
results
    while(residual > TOL):
        beta = 1 - alpha*np.dot(e, Q @ r)
        z = alpha*(Q @ r) + beta*v
        residual = np.linalg.norm(r-z, ord=1)
```

```
        r = z
        loop_count += 1
    print(f'power method converged in {loop_count} iterations')
    return r, loop_count

def get_Pagerank(Q, n, alpha, r_init, TOL):
    P = get_P_matrix(Q, n)
    # P_tester.draw_graph()
    # G = get_Google_matrix(P, alpha, n)
    return power_method(Q, alpha, r_init, n, TOL)

def test_PR_results(r, Q, n, alpha, rTOL):
    P = get_P_matrix(Q, n)
    G = get_Google_matrix(P, alpha, n)
    print(f'Pagerank vector: r = \n{r}')
    print(f'Google matrix times Pagerank vector: Gr =\n{G @ r}')
    print(f'Gr = r? \t->\t{np.allclose(r, G @ r, rtol=rTOL)}\n')

if __name__ == '__main__':
    # Problem 7.3
    Q = np.array([[0,1/3,0,0,0,0],
                  [1/3,0,0,0,0,0],
                  [0,1/3,0,0,1/3,1/2],
                  [1/3,0,0,0,1/3,0],
                  [1/3,1/3,0,0,0,1/2],
                  [0,0,1,0,1/3,0]])
    # damping factor in (0,1) according to Brin and Page (1996)
    alpha = 0.85
    r_TOL = 10e-4
    n = Q.shape[0] # Q must be square
    r_init = np.random.rand(n)
    r_init /= np.linalg.norm(r_init, ord=1)

    print('Problem 7.3:')
    r, num_iter1 = get_Pagerank(Q, n, alpha, r_init, r_TOL)
    test_PR_results(r, Q, n, alpha, r_TOL)

    # Problem 7.4
    Q_tilde = deepcopy(Q)
    # new outlink from 4 to 5 means an update to column 4, row 5 only since 4 had no
outlinks
    Q_tilde[4,3] = 1
    print('Problem 7.4:')
    r_tilde, num_iter2 = get_Pagerank(Q_tilde, n, alpha, r_init, r_TOL)
    test_PR_results(r_tilde, Q_tilde, n, alpha, r_TOL)
```

[redacted]