Assignment 3: Eigenvectors, eigenvalues, Google PageRank

Working on the assignment, todo and not todo:

Todo:

- ✓ **Work by yourself** and submit your own assignment, **no pairing** to other students
- ✓ Test and save your assignment - **submit the last tested and saved version**
- ✓ to **submit** the assignment, download the notebook (File → Download .ipynb in Google Colab)
- ✓ submit **only** the **ipynb** file under the name **hw3.ipynb**
- ✓ It is advisable to add extra cells to check your code implementation

Not todo:

- ✗ do **NOT** submit an **empty assignment**
- ✗ do **NOT submit extra files**, unless you're asked to do so
- ✗ **Do NOT submit a .py/.txt/.rar/.zip (or any non (.ipynb) file)** version for the notebook of the assignment
- ✗ **do NOT change** the notebook file name

```
import numpy as np
import numpy.linalg as la
import networkx as nx
import matplotlib.pyplot as plt
```

Question 1: eigenvalues and eigenvectors

Matrix $M = \begin{pmatrix} -3 & -2 & 7 \\ 1 & 6 & 1 \\ 3 & 6 & -1 \end{pmatrix}$ is given.

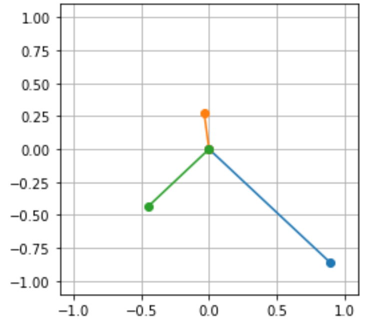
Q1.1 Find eigenvalues and eigenvectors of the matrix.

```
Implement the function eigM() that returns a tuple (Eigenvalues, Eigenvectors)
------------
Input parameters:
-nothing-
------------
return value:
- Eigenvalues - list of the eigenvalues of the matrix M
- Eigenvectors - list of the eigenvectors of the matrix M
------------
```

```
def eigM():
  pass
```

```
# --------------------------- RUN THIS TEST CODE CELL -------------------------------------
# Q1.1 --- Test your implementation:
# ---------------------------
print ("Test - Testing the implementation of the 'eigM' function..\n ")
title = "Eigenvalues and eigenvectors of the given matrix:"
l,v = eigM()
print(f"\n {title} ")
print("="*(len(title)+2))
print(f"Eigenvalues: {l}")
print(f"Eigenvectors:\n{v}")
print ("\n\n there will be hidden tests ... ")
```

Q1.2 The eigenvectors are three-dimensional. Plot projections of eigenvectors on each of 2-dimensional spaces. Hint: a three-dimensional vector (x_1, x_2, x_3) has 3 two-dimensional projections: (x_1, x_2), (x_1, x_3), (x_2, x_3).
An example for one of the three plotting will be:




```
def vectorsPlot (vec,x ,y): # You can use this helper function
  plt.figure(figsize=(4, 4))
  plt.xlim((-1.1, 1.1))
  plt.ylim((-1.1, 1.1))
  plt.grid()
  for vector in vec:
    plt.plot([0,vector[0, x]], [0,vector[0, y]], '-o')
  plt.show()

# Write your code here (you dont need to define a function)
```

Q1.3 Find a matrix with the same eigenvectors but with eigenvalues of half the value of M's eigenvalues.

```
Implement the function halfEigenM() that returns the requested Matrix above
------------
Input parameters:
```

```
-nothing-
------------
return value:
- A - requested Matrix
------------
```

```
def halfEigenM():
    pass
```

```
# --------------------------- RUN THIS TEST CODE CELL -------------------------------------
# Q1.3 --- Test your implementation:
# ---------------------------
print ("Test - Testing the implementation of the 'halfEigenM' function..\n ")
print(f"new Matrix:\n {halfEigenM()}")
print ("\n\n there will be hidden tests ... ")
```

Q1.4. Find a matrix with the same eigenvalues as M from Q1.1 but different eigenvectors.

```
compute diffVectorsM() that returns the requested Matrix above
------------
Input parameters:
-nothing-
------------
return value:
- M - requested Matrix
------------
```

```
def diffVectorsM():
   pass
```

```
# --------------------------- RUN THIS TEST CODE CELL -------------------------------------
# Q1.4 --- Test your implementation:
# ---------------------------
print ("Test - Testing the implementation of the 'diffVectorsM' function..\n ")
print(f"new Matrix:\n {diffVectorsM()}")
print ("\n\n there will be hidden tests ... ")
```

Q1.5 Write a function that given a list of eigenvalues, returns a matrix with those eigenvalues. Example:

```
>> eigenvalues_to_matrix([0, 2])
[[1, -1],
 [-1, 1]]
```

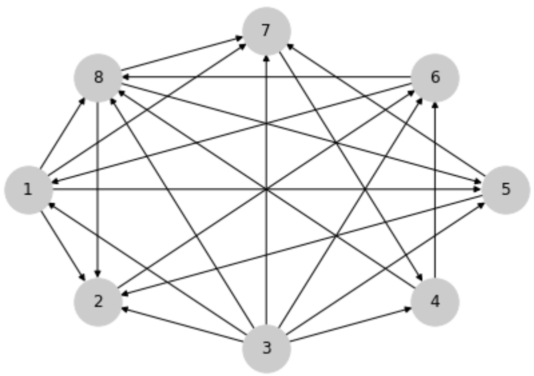
```
Implement the function eigenvalues_to_matrix(eigenvalues) satisfying the requirments above
------------
Input parameters:
- eigenvalues - list of eigenvalues
------------
return value:
- M - requested Matrix
------------
```

```
def eigenvalues_to_matrix(eig_vals):
   pass
```

```
# --------------------------- RUN THIS TEST CODE CELL -------------------------------------
# Q1.5 --- Test your implementation:
# ---------------------------
print ("Test - Testing the implementation of the 'eigenvalues_to_matrix' function..\n ")
print(f"new Matrix:\n {eigenvalues_to_matrix([8, 7, 6])}")
print ("\n\n there will be hidden tests ... ")
```

Question 2: Google PageRank

A directed graph G is given:



Q2.1 Write adjacency matrix A for G.

```
Implement adj_matrix() according to the image above
------------
Input parameters:
- nothing -
```

```
------------
return value:
- M - requested Matrix
------------
```

```
def adj_matrix():
   pass
```

```
# --------------------------- RUN THIS TEST CODE CELL -------------------------------------
# Q2.1 --- Test your implementation:
# ---------------------------
print ("Test - Testing the implementation of the 'adj_matrix' function..\n ")
print(f"the adjacency matrix:\n {np.array(adj_matrix())}")
print ("\n\n there will be hidden tests ... ")
```

Q2.2 Compute Google PageRank of all nodes in G.

Use the iterative solution

```
Implement PG_rank_iter(M) that returns a vector of pagerank of all nodes of Matrix (for any matrix)
------------
Input parameters:
- M - Matrix to apply pagerank to
- d - Damping factor (default value = 0.85 as seen in class)
------------
return value:
- pagerank_vector - pagerank vector of all the nodes
------------
```

```
def PG_rank_iter(M,d=0.85):
    pass
```

```
# --------------------------- RUN THIS TEST CODE CELL -------------------------------------
# Q2.2 --- Test your implementation:
# ---------------------------
print ("Test - Testing the implementation of the 'PG_rank_iter' function..\n ")
M = adj_matrix()
va = PG_rank_iter(M)
for i in range(len(va)):
    print("Node no.",str(i+1),", google pagerank=",va[i])
print ("\n\n there will be hidden tests ... ")
```

Q2.3 Change the graph edges (in the adjacency matrix of Q2.1) such that node 1 has the highest rank.

Use the algebraic solution

```
Implement n1_highest_Rank() that returns a vector of pagerank of all nodes of the Matrix after the change
------------
Input parameters:
- Nothing -
------------
return value:
- pagerank_vector - pagerank vector of all the nodes
------------
```

```
def n1_highest_Rank():
    pass
```

```
# --------------------------- RUN THIS TEST CODE CELL -------------------------------------
# Q2.3 --- Test your implementation:
# ---------------------------
print ("Test - Testing the implementation of the 'n1_highest_Rank' function..\n ")
va = n1_highest_Rank()
for i in range(len(va)):
    print("Node no.",str(i+1),", google pagerank=",va[i])
print ("\n\n there will be hidden tests ... ")
```

Q2.4 Write a function that inverts the directions of all edges of a graph, given its adjacency matrix M

```
Write invert_graph(M) that returns the adjacency matrix of the inverted graph
------------
Input parameters:
- M - Adjacency matrix of some graph
return value:
- M_inv - Adjacency matrix of the inverted graph
------------
```

```
def invert_graph(M):
    pass
```

```
# --------------------------- RUN THIS TEST CODE CELL -------------------------------------
# Q2.4 --- Test your implementation:
# ---------------------------
print ("Test - Testing the implementation of the 'invert_graph' function..\n ")
print(f"the inverted graph adjacency matrix is :\n {invert_graph(np.array(adj_matrix()))}")
print ("\n\n there will be hidden tests ... ")
```

Q2.5 How many iterations of iterative PageRank algorithm are required to approximate the exact (algebraic) PageRank algorithm with less than 0.025% error? Hint: use L1 norm `la.norm(vector, 1)` to determine the relative error between v_alg and v_iter.

$$\text{relative error} = \frac{||v_{alg} - v_{iter}||}{||v_{alg}||}$$

For reading more about norms: https://en.wikipedia.org/wiki/Norm_(mathematics)

```
Write PageRank_iterative_by_percent(M, precent, d) that returns the number of iterations need for less than 0.025% error
------------
Input parameters:
- M -   numpy array
        adjacency matrix

- percent -   float
        error by percentages

- d -   float, optional
        damping factor, by default 0.85


return value:
- answer - int
            number of iterations to meet the desired percentages
------------
```

```
def PageRank_iterative_by_percent(M,percent,d=0.85):
    pass
```

```
# --------------------------- RUN THIS TEST CODE CELL -------------------------------------
# Q2.5 --- Test your implementation:
# ---------------------------
print ("Test - Testing the implementation of the 'PageRank_iterative_by_percent' function..\n ")
percentage = 0.025 # 0.025%
print(f"the itertive PageRank algorithm need to run :   {PageRank_iterative_by_percent(adj_matrix(),percentage)} iterartions")
print ("\n\n there will be hidden tests ... ")
```