محدثه حکیمی – ۴۰۱۲۴۳۰۴۵
گزارش تمرین ۳ کامپیوتری

سؤال اول)

بخش الف – توی این قسمت باید با استفاده از روش توانی بردار ویژه‌ی متناظر با بزرگترین مقدار ویژه‌ی ماتریس رو به دست بیاریم و با توجه به مفهوم مرکزیت بردار ویژه سه نود از گراف رو که بیشترین ارتباط با بقیه رو دارن برگردونیم.

اول از همه تابعی که بردار ویژه رو با روش توانی به دست میاره:

```python
convergence_criteria = 0.000001 # 10^-6


def find_eigenvectors_using_power_iteration(matrix, iterations=1000):
    x = np.array([1 for i in range(len(matrix))])   # initial guess
    for i in range(iterations):
        x_prev = x.copy()
        x = np.dot(matrix, x)
        x = x / np.linalg.norm(x)

        # check for convergence
        if np.abs(np.linalg.norm(x) - np.linalg.norm(x_prev)) < convergence_criteria:
            return x
    print(f"Did not converge in {iterations} iterations")
    return x
```

اینجا درواقع از این فرمول باید استفاده بشه:

روش توانی :

$$X_i = A X_{i-1} \qquad i = 1, 2, \dots$$

X_0 حدس اولیه .

$$X_1 = A X_0$$

$$X_2 = A X_1 = A^2 X_0$$

$$\vdots$$

$$X_m = A^m X_0$$

اگر m بزرگ باشد، آنگاه X_m یک تخمین از بردار ویژه غالب است.

حدس اولیه رو یک بردار با مقادیر تماماً یک گرفتم. میزان iteration هم ۱۰۰۰ئه. و هر بار که ماتریس رو در X ضرب می‌کنم نتیجه رو یکه می‌کنم.

یک معیار همگرایی هم اون بالا تعریف شده که مقدارش 10^{-6} هست. در هر بار پیمایش چک می‌کنیم که آیا اندازه‌ی تخمین فعلی (X) از تخمین پیمایش قبلی (x_prev) کوچکتر از معیار همگرایی شده یا نه. به عبارتی آیا به جایی از پیمایش رسیدیم که میزان تغییرات ناچیز باشن؟ در این صورت حلقه رو می‌شکنیم و X رو برمی‌گردونیم. اگر این شرط هیچ وقت درست نباشه هم یعنی که X همگرا نشده و همون مقدار رو برمی‌گردونیم.

حالا باید ۳ نود مرکزی رو به دست بیاریم:

```python
def find_top_three_nodes_for_centrality(matrix):
    # find eigenvector corresponding to largest eigenvalue
    eigenvector = find_eigenvectors_using_power_iteration(matrix)
    # find top 3 nodes
    centrality = np.abs(eigenvector)
    top_three_nodes = np.argsort(centrality)[::-1][:3]
    return top_three_nodes

print("Eigenvector corresponding to largest eigenvalue: ")
print(find_eigenvectors_using_power_iteration(matrix))
print("Top 3 nodes with the highest centrality: ")
print(find_top_three_nodes_for_centrality(matrix))
```

اول از همه باید مفهوم مرکزیت رو بفهمیم. مرکزیت اینجا یعنی هر نود چه میزان با نودهای دیگه در ارتباطه و درواقع میزان connected بودن نودها رو می‌سنجیم. و در بردار ویژه‌ی به دست اومده هر درایه میزان مرکزیت (centrality score) نود مربوطه رو نشون می‌ده.

پس نودها رو بر اساس مقدار درایه‌ها سورت می‌کنیم و ۳تای اول رو برمی‌گردونیم، داخل کد هم مشهوده.

برای تست هم ماتریس زیر رو ورودی دادم و خروجی:

```python
matrix = np.array(
    [[0, 0.1, 0.5, 0.3, 0.7],
     [0.1, 0, 0.2, 0.4, 0.8],
     [0.5, 0.2, 0, 0.9, 0.6],
     [0.3, 0.4, 0.9, 0, 0.3],
     [0.7, 0.8, 0.6, 0.3, 0]]
)
```

```
Eigenvector corresponding to largest eigenvalue:
[0.40941021 0.38367586 0.49714097 0.44216303 0.492462  ]
Top 3 nodes with the highest centrality:
[2 4 3]
```

که به ترتیب نودهای ۲ و ۴ و ۳ رو برگردونده. با دقت در مقادیر ماتریس هم می‌شه دید که این ۳ نود بیشترین ارتباط با دیگر نودهای گراف رو دارن.

ب – در این قسمت باید مشخص کنیم که برای هر دو عضو دانشکده، می‌توان تعیین کرد رابطه‌ی دوستی وجود داره یا نه. پس باید گراف رو از لحاظ connectivity بررسی کرد و اگر A در ارتباط نزدیکی با B بود، اون وقت اونها رو با هم دوست اعلام کرد.

اینجا ماتریس لاپلاسین به کمک ما میاد. این ماتریس از دو ماتریس مجاورت و درجه به دست میاد و فرمولش: $L = D - A$. هر درایه از این ماتریس اختلاف درجه‌ی بین نودهای متصل به هم رو نشون می‌ده و در حالت کلی، به ما می‌گه که روابط نودهای گراف به چه صورته.

همچنین یک ویژگی مهم این ماتریس Fiedler Vector هست که درواقع بردار ویژه‌ی متناظر با second smallest eigenvalue هست. این بردار هم خاصیت‌های مهمی داره از جمله اینکه نودهایی که در اون مقادیر نزدیکی دارن، به احتمال بالایی در یک گروه یا cluster در گراف اصلی هستن. ما در اینجا با توجه به علامت‌های هر کدوم از نودها پیش می‌ریم. داخل کد نشون می‌دم.

اول از همه دوتا تابع برای ماتریس لاپلاسین و Fiedler vector می‌نویسیم:

```python
# construct a laplacian matrix from the adjacency matrix
def construct_laplacian_matrix(matrix):
    degree_matrix = np.diag([sum(matrix[i]) for i in range(len(matrix))])
    laplacian_matrix = degree_matrix - matrix
    return laplacian_matrix

# find Fiedler vector (the second-smallest eigenvector) of the Laplacian matrix
def find_fiedler_vector(matrix):
    laplacian_matrix = construct_laplacian_matrix(matrix)
    eigenvalues, eigenvectors = np.linalg.eig(laplacian_matrix)
    fiedler_vector = eigenvectors[:, 1]
    return fiedler_vector
```

و یک تابع هم برای جداکردن clusterها داریم که بر اساس مثبت و منفی‌بودن نودها، گروه‌های دوستی رو از هم جدا می‌کنه (اینجا می‌تونیم یه کلاستر جدا برای مقادیر مساوی با صفر هم در نظر بگیریم):

```python
def find_clusters_of_friendship(matrix):
    fiedler_vector = find_fiedler_vector(matrix)
    # separate clusters based on the sign of the Fiedler vector
    cluster1 = [i for i in range(len(fiedler_vector)) if fiedler_vector[i] >= 0]
    cluster2 = [i for i in range(len(fiedler_vector)) if fiedler_vector[i] < 0]
    return cluster1, cluster2
```

به طور مثال برای ماتریسی که بالاتر به برنامه دادیم:

```
Laplacian matrix:
[[ 1.6 -0.1 -0.5 -0.3 -0.7]
 [-0.1  1.5 -0.2 -0.4 -0.8]
 [-0.5 -0.2  2.2 -0.9 -0.6]
 [-0.3 -0.4 -0.9  1.9 -0.3]
 [-0.7 -0.8 -0.6 -0.3  2.4]]
Fiedler vector:
[-0.17243677 -0.31934463 -0.52996656  0.32999342  0.69175455]
```

و گروه‌های دوستی:

```
Fiedler vector:
[-0.17243677 -0.31934463 -0.52996656  0.32999342  0.69175455]
Clusters of friendship:
([3, 4], [0, 1, 2])
```

به این ترتیب ۳ و ۴، با هم دوستن و ۰ و ۱ و ۲ با هم.

اما نکته‌ی دیگه‌ای که در مورد ماتریس لاپلاسین وجود داره اینه که nullity ماتریس، برابره با تعداد componentهایی که گراف داره. به عبارتی بهمون می‌گه که گراف چندپاره‌ست.

Nullity رو از فرمول رو به رو به دست آوردم و برای ماتریس داده شده:

```python
def nullity(matrix):
    return len(matrix) - np.linalg.matrix_rank(matrix)
```

```
Nullity of the Laplacian matrix:  1
```

که یعنی تمام نودهای گراف به هم وصلن که با ماتریس ورودی همخوانی داره.

سؤال دوم) برای این سؤال باید عدد nام فیبوناچی رو به کمک ماتریس به دست بیاریم ($n = 2 ^ 100 + 1234$). روش اول به توان رسوندن مستقیم ماتریسه که چون باید در $\log n$ انجام بشه، از قانون $A^2 = A * A$ استفاده می‌کنیم و به طور بازگشتی A^n رو به دست میاریم.

```python
import numpy as np
import time


N = 2 ** 100 + 1234
fib_matrix = np.array([[1, 1], [1, 0]])


def matrix_multiplication(matrix, n):
    # use A^2 to calculate in O(log n)
    if n == 1:
        return matrix
    else:
        half = matrix_multiplication(matrix, n//2)
        if n % 2 == 0:
            return np.dot(half, half)
        else:
            return np.dot(np.dot(half, half), matrix)
```

نتیجه‌ی اجرای کد:

```
calculating A^N using matrix multiplication in O(log n)...
result:
[[6540800425176545154 3722172121269335201]
 [3722172121269335201 2818628303907209953]]
nth fibonacci number:  3722172121269335201
Time taken:  0.0005271434783935547 seconds
```

که همونطور که مشخصه به وضوح به اورفلو خورده چون مقدار فیبوناچی رو کوچکتر از ورودی به دست آورده.

روش دوم هم با استفاده از بردارهای ویژه است.

فرمولی که ازش استفاده کردم این بود:

$$A = PDP^{-1} \quad ==> A^n = PD^nP^{-1}$$

که P ماتریس متشکل از بردارهای ویژه‌ی A و D ماتریس قطری مقادیر ویژه‌ی A هست.

تساوی سمت راست هم از اینجا نتیجه‌گیری می‌شه که $I = P * P^{-1}$ و در ضرب P با وارونش خط می‌خوره.

کد:

```python
def matrix_multiplication_using_eigenvectors(matrix):
    # A = PDP^-1
    # A^n = PD^nP^-1

    eigenvalues, eigenvectors = np.linalg.eig(matrix)
    P = eigenvectors
    # D = np.diag(eigenvalues)
    P_inv = np.linalg.inv(P)
    # use matrix_multiplication to calculate D^n
    D_power_n = matrix_multiplication(np.diag(eigenvalues), N)
    result = np.dot(np.dot(P, D_power_n), P_inv)
    # if the result is in scientific notation, convert it to an i
    # result = np.array([[int(result[i][j]) for j in range(len(re
    return result
```

نتیجه‌ی اجرای کد:

```
calculating A^N using eigenvectors...
result:
[[nan nan]
 [nan nan]]
nth fibonacci number:  nan
Time taken:  0.0012726783752441406 seconds
```

که بازم به دلیل بزرگی N اورفلو کرده و کلاً Nan برگردونده.

(توی پرانتز بگم که من از هر روشی که می‌شد رفتم برای محاسبه‌ی D^n، از math.pow و عملگر ** و هر راه دیگه‌ای که بود، و در نهایت یا ارور می‌خورد یا هم اورفلو داشت. دیگه به همین روش بسنده کردم، یعنی از تابع بالایی که محاسبه‌ی توان از اردر لاگ بود استفاده کردم.)

برای اینی که مطمئن بشم کد درسته هم یه ورودی دیگه (n = 30) بهش دادم و خروجی:

```
calculating A^N using matrix multiplication in O(log n)...
result:
[[1346269  832040]
 [ 832040  514229]]
nth fibonacci number:  832040
Time taken:  0.00021886825561523438 seconds
```

```
calculating A^N using eigenvectors...
result:
[[1346269.  832040.]
 [ 832040.  514229.]]
nth fibonacci number:  832040.0000000001
Time taken:  0.0007274150848388672 seconds
```

برای زمان محاسبه هم می‌شه دید که روش بردارهای ویژه چندبرابر روش دیگه‌ست.

پ.ن: یه خط کد کامنت‌شده داخل تابع محاسبه با بردارهای ویژه هست که صرفاً درایه‌ها رو از نوتیشن علمی به عدد صحیح درمیاره. برای مقادیر بزرگ‌تر n و مقایسه‌ش با نتیجه‌ی تابع دیگه لازم بود.

اصلاحیه: برای n=1234 کد رو دوباره اجرا کردم و نتیجه:

```
calculating A^N using eigenvectors...
result:
[[5.62666043e+257 3.47746739e+257]
 [3.47746739e+257 2.14919304e+257]]
nth fibonacci number:  3.477467391803717e+257
Time taken:  0.0020589828491210938 seconds
```

```
calculating A^N using matrix multiplication in O(log n)...
result:
[[ 2654805133464336809 -2651754408573296569]
 [-2651754408573296569  5306559542037633378]]
nth fibonacci number:  -2651754408573296569
Time taken:  0.001085042953491211 seconds
```

که نتیجه‌ی سمت چپی اورفلو کرده و کد سمت راست هم بعد از تبدیل به عدد صحیح اینطور می‌شه:

```
result:
[[562666043470786140987014004364455922830291254449518195833126492124477401145175508196761925338604020403328657564246422586816366843722311490696160201122378809106134779388840824752807502381465943413687931312079272298471628420463629239731177204418909236264697856

347746739180371690203199316950597274580436062820877071858230443699913266674995758441600205448424256303143556282093284667455391092236800895219882852751938327474531633940082949609141922938432167022413433847350502063113724778470905985954988267613739686916259840]
 [347746739180371636859083853911763081941621737050814694700163865172647983504933264329302302907367684131685949309658958724016106344770877328109469619797017684567683215293442580847881286826315699317787146552838304231393595691378090132429003967802049357605240832

214919304290414397439699224375024455611040865858578746816829469897298851300117255642863817349123191928727494309718811975921691004019587028365864115415519838724754726802117506382404943330917308686648210170216572403637774554899907400250204636993479220037419008]]
```

که عدد فیبوناچی می‌شه عدد دومی از بالا.

زمان اجرای روش بردارهای ویژه باز هم بیشتر از روش ضرب ماتریسه.