Problem 1: Desarrollar un programa que calcule los m valores propios y vectores propios más grandes para matrices de tamaño n mediante el **método de la potencia. (2 puntos)**

Solution. El código para resolver este problema es el mismo que se utilizó en la tarea pasada. Ahora, adicionalmente presentaremos los vectores propios que se consiguen en el proceso del método.
Ejemplos:

1. **Eigen_3x3.txt**

$$\begin{bmatrix} 5.0 & -1.778 & 0.0 \\ -1.778 & 9.0 & -1.778 \\ 0.0 & -1.778 & 10.0 \end{bmatrix}$$

Como vector inicial, usaremos $\mathbf{x}_0^i = 3^{-1/2}$ para $1 \le i \le 3$. Ejecutando nuestro programa con

```
make run-p1 ARGS="t5a/Eigen_3x3.txt t5a/x0_3.txt 3 1e-12 10000
              c_eigenvalues.txt c_eigenvectors.txt",
```

obtenemos los 3 eigenvalores más grandes (en este caso, todos). Para verificar, usamos `numpy`. Corremos el programa de `python` con

```
python3 p1_verify.py t5a/Eigen_3x3.txt 3 1
```

Eigenvectores	
Numpy	M. de Potencia
$[0.175\ \ -0.644\ \ 0.745]$	$[0.175\ \ -0.644\ \ 0.745]$
$[-0.365\ \ 0.660\ \ 0.656]$	$[0.365\ \ -0.660\ \ -0.656]$
$[0.914\ \ 0.386\ \ 0.119]$	$[-0.914\ \ -0.386\ \ -0.119]$

Table 1: Comparación de resultados de Eigen_3x3.txt

Eigenvalores	
Numpy	M. de Potencia
11.538405	11.538403
8.213809	8.213811
4.247786	4.247786

Table 2: Comparación de eigenvalores de Eigen_3x3.txt

2. **Eigen_5x5.txt**

$$\begin{bmatrix} 0.2 & 0.1 & 1 & 1 & 0 \\ 0.1 & 4 & -1 & 1 & -1 \\ 1 & -1 & 60 & 0 & -2 \\ 1 & 1 & 0 & 8 & 4 \\ 0 & -1 & -2 & 4 & 700 \end{bmatrix}$$

Como vector inicial, usaremos $\mathbf{x}_0^i = 5^{-1/2}$. Ejecutando nuestro programa con

```
make run-p1 ARGS="t5a/Eigen_5x5.txt t5a/x0_5.txt 3 1e-12 10000
              c_eigenvalues.txt c_eigenvectors.txt",
```

obtenemos los 3 eigenvalores más grandes. Verificamos los resultados con

```
python3 p1_verify.py t5a/Eigen_5x5.txt 3 1
```

Eigenvectores	
Numpy	M. de Potencia
$[0.000\ \ -0.001\ \ -0.003\ \ 0.005\ \ 0.999]$	$[-0.000\ \ 0.001\ \ 0.003\ \ -0.005\ \ -0.999]$
$[0.016\ \ -0.017\ \ 0.99\ \ 0.0002\ \ 0.003]$	$[-0.016\ \ 0.017\ \ -0.999\ \ -0.0002\ \ -0.003]$
$[0.122\ \ 0.226\ \ 0.002\ \ 0.966\ \ -0.005]$	$[0.111\ \ 0.207\ \ -0.003\ \ 0.886\ \ 0.399]$

Table 3: Comparación de eigenvectores de Eigen_5x5.txt

Eigenvalores	
Numpy	M. de Potencia
700.030781	700.030781
60.028366	60.028366
8.338447	8.338448

Table 4: Comparación de eigenvalores de Eigen_5x5.txt

Podemos observar muy poca diferencia entre los eigenvalores, por lo que podemos asumir que han sido verificados como correctos.

3. **Eigen_50x50.txt**

$$\begin{bmatrix} 10.000 & 0.084 & 0.039 & \cdots & 0.052 & 0.078 & 0.007 \\ 0.084 & 20.000 & 0.095 & \cdots & 0.040 & 0.053 & 0.040 \\ 0.039 & 0.095 & 30.000 & \cdots & 0.084 & 0.013 & 0.052 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0.081 & 0.044 & 0.095 & \cdots & 380.000 & 0.038 & 0.081 \\ 0.092 & 0.092 & 0.005 & \cdots & 0.041 & 490.000 & 0.092 \\ 0.007 & 0.040 & 0.052 & \cdots & 0.078 & 0.078 & 500.000 \end{bmatrix}$$

Como vector inicial, usaremos $\mathbf{x}_0^i = 50^{-1/2}$ Ejecutando nuestro programa con

```
make run-p1 ARGS="t5a/Eigen_50x50.txt t5a/x0_50.txt 7 1e-12 10000
              c_eigenvalues.txt c_eigenvectors.txt",
```

obtenemos los 7 eigenvalores más grandes. Para verificar el resultado, usar

```
python3 p1_verify.py t5a/Eigen_50x50.txt 7 1
```

Eigenvalores	
Numpy	M. de Potencia
500.001543	500.001591
490.001441	490.001430
480.000527	480.000520
470.001058	470.001067
460.001269	460.001265
450.000483	450.000481
440.000419	440.000418

Table 5: Comparación de eigenvectores de Eigen_50x50.txt

Para ver las normas de $\|Ax - \lambda x\|$, obtenemos el archivo `verify.txt`, que contiene los valores

n	$\|Ax - \lambda x\|$
1	1.5159×10^{-5}
2	2.1505×10^{-5}
3	2.1678×10^{-5}
4	2.1239×10^{-5}
5	2.0382×10^{-5}
6	2.0829×10^{-5}
7	2.0991×10^{-5}

Table 6: Norma residual $\|Ax - \lambda x\|$

4. **Eigen_125x125.txt**

$$\begin{bmatrix} 9.566e-05 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1.965e-04 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 3.802e-04 & 1.356e-06 & 0 \\ 0 & 0 & 0 & \cdots & 1.356e-06 & 3.701e-04 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 3.522e-04 \end{bmatrix}$$

Como vector inicial, usaremos $\mathbf{x}_0^i = 125^{-1/2}$, Ejecutando nuestro programa con

```
make run-p1 ARGS="make run-p1 ARGS="t5a/Eigen_125x125.txt t5a/x0_125.txt 7
              1e-12 10000 c_eigenvalues.txt c_eigenvectors.txt",
```

```
python3 p1_verify.py t5a/Eigen_125x125.txt 7 1
```

obtenemos los 7 eigenvalores más grandes. Notemos que en la siguiente tabla, comparamos los eigenvalores distintos. El étodo de la potencia, no obtenemos los 12 eigenvalores igual a 1, pero si obtenemos los 7 eigenvalores más grandes distintos.

Eigenvalores	
Numpy	M. de Potencia
1.000000000000000	1.000000000000000
0.000597537881445	0.000597537379115193
0.000580571976900	0.000580571970964
0.000561184341205	0.000561184198043
0.000556494725919	0.000556402899526
0.000550804423138	0.000551046077159
0.000547941791866	0.000547801563565

Table 7: Comparación de eigenvalores de Eigen_125x125.txt

n	$\|Ax - \lambda x\|$
1	5.0×10^{-15}
2	1.0×10^{-15}
3	1.0×10^{-15}
4	2.0×10^{-15}
5	2.0×10^{-15}
6	2.0×10^{-15}
7	4.0×10^{-15}

Table 8: Norma residual $\|Ax - \lambda x\|$

□

Problem 2: Desarrollar un programa que calcule los m valores propios y vectores propios más pequeños para matrices de tamaño n mediante el **método de la potencia inversa. (2 puntos)**

Solution.

1. **Eigen_3x3.txt**

$$\begin{bmatrix} 5.0 & -1.778 & 0.0 \\ -1.778 & 9.0 & -1.778 \\ 0.0 & -1.778 & 10.0 \end{bmatrix}$$

Como vector inicial, usaremos $\mathbf{x}_0^i = 3^{-1/2}$ Ejecutando nuestro programa con

```
make run-p2 ARGS="t5a/Eigen_3x3.txt t5a/x0_3.txt 3 1e-12 10000
              c_eigenvalues.txt c_eigenvectors.txt",
```

obtenemos los 3 eigenvalores más pequeños (en este caso, todos).

```
python3 p1_verify.py t5a/Eigen_3x3.txt 3 0
```

Eigenvectores	
Numpy	M. de Potencia
$[-0.914\ \ -0.387\ \ -0.120]$	$[0.914\ \ 0.387\ \ 0.120]$
$[0.365\ \ -0.660\ \ -0.657]$	$[-0.365\ \ 0.660\ \ 0.657]$
$[0.175\ \ -0.644\ \ 0.745]$	$[0.175\ \ -0.644\ \ 0.745]$

Table 9: Comparación de resultados de Eigen_3x3.txt

Eigenvalores	
Numpy	M. de Potencia
4.247785692735033	4.247785692735150
8.213809393221599	8.213809393222336
11.538404914043371	11.538404914042516

Table 10: Comparación de eigenvalores de Eigen_3x3.txt

2. **Eigen_5x5.txt**

$$\begin{bmatrix} 0.2 & 0.1 & 1 & 1 & 0 \\ 0.1 & 4 & -1 & 1 & -1 \\ 1 & -1 & 60 & 0 & -2 \\ 1 & 1 & 0 & 8 & 4 \\ 0 & -1 & -2 & 4 & 700 \end{bmatrix}$$

Como vector inicial, usaremos $\mathbf{x}_0 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$ Ejecutando nuestro programa con

```
make run-p2 ARGS="t5a/Eigen_5x5.txt t5a/x0_5.txt 3 1e-12 10000
              c_eigenvalues.txt c_eigenvectors.txt",
```

```
python3 p1_verify.py t5a/Eigen_5x5.txt 3 0
```

obtenemos los 3 eigenvalores más pequeños. Como verif

Eigenvectores	
Numpy	M. de Potencia
$[0.992\ \ 0.003\ \ -0.016\ \ -0.126\ \ 0.001]$	$[0.992\ \ 0.003\ \ -0.016\ \ -0.126\ \ 0.001]$
$[-0.031\ \ 0.974\ \ 0.018\ \ -0.224\ \ 0.003]$	$[-0.031\ \ 0.974\ \ 0.018\ \ -0.224\ \ 0.003]$
$[0.122\ \ 0.226\ \ 0.002\ \ 0.966\ \ -0.005]$	$[0.122\ \ 0.226\ \ 0.002\ \ 0.966\ \ -0.005]$

Table 11: Comparación de resultados de Eigen_5x5.txt

Eigenvalores	
Numpy	M. de Potencia
9.998050174885066	9.998050174885281
19.998793857960756	19.998793857961047
29.998930953123583	29.998930953123931
39.999763292782198	39.999763292782603
49.998366378355584	49.998366378356302
59.999844677982182	59.999844677982736
69.999135451279130	69.999135451279898

Table 12: Comparación de eigenvalores de Eigen_5x5.txt

3. **Eigen_50x50.txt**

$$\begin{bmatrix} 10.000 & 0.084 & 0.039 & \cdots & 0.052 & 0.078 & 0.007 \\ 0.084 & 20.000 & 0.095 & \cdots & 0.040 & 0.053 & 0.040 \\ 0.039 & 0.095 & 30.000 & \cdots & 0.084 & 0.013 & 0.052 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0.081 & 0.044 & 0.095 & \cdots & 380.000 & 0.038 & 0.081 \\ 0.092 & 0.092 & 0.005 & \cdots & 0.041 & 490.000 & 0.092 \\ 0.007 & 0.040 & 0.052 & \cdots & 0.078 & 0.078 & 500.000 \end{bmatrix}$$

Como vector inicial, usaremos $\mathbf{x}_0 = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}^T$ Ejecutando nuestro programa con

```
make run-p2 ARGS="t5a/Eigen_50x50.txt t5a/x0_50.txt 7 1e-12 10000
              c_eigenvalues.txt c_eigenvectors.txt",
```

```
python3 p1_verify.py t5a/Eigen_50x50.txt 7 0
```

obtenemos los 7 eigenvalores más pequeños.

Eigenvalores	
Numpy	M. de Potencia
9.998050174885066	9.998050174885281
19.998793857960756	19.998793857961047
29.998930953123583	29.998930953123931
39.999763292782198	39.999763292782603
49.998366378355584	49.998366378356302
59.999844677982182	59.999844677982736
69.999135451279130	69.999135451279898

Table 13: Comparación de eigenvalores

Norma del residuo	
n	$\|Ax - \lambda x\|$
1	1.201×10^{-6}
2	2.362×10^{-6}
3	3.804×10^{-6}
4	5.104×10^{-6}
5	6.145×10^{-6}
6	6.902×10^{-6}
7	7.463×10^{-6}

Table 14: Norma del residuo $\|Ax - \lambda x\|$ para cada vector n

4. **Eigen_125x125.txt**

$$\begin{bmatrix} 9.566e-05 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1.965e-04 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 3.802e-04 & 1.356e-06 & 0 \\ 0 & 0 & 0 & \cdots & 1.356e-06 & 3.701e-04 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 3.522e-04 \end{bmatrix}$$

Como vector inicial, usaremos $\mathbf{x}_0 = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \end{bmatrix}^T$ Ejecutando nuestro programa con

```
make run-p2 ARGS="t5a/Eigen_125x125.txt t5a/x0_125.txt 7 1e-12 10000
              c_eigenvalues.txt c_eigenvectors.txt",
```

```
python3 p1_verify.py t5a/Eigen_125x125.txt 7 0
```

obtenemos los 7 eigenvalores más pequeños.

Eigenvalores	
Numpy	M. de Potencia
0.000001862755663	0.000001862755675
0.000008134856129	0.000008134856232
0.000016501222866	0.000016501229475
0.000017624976804	0.000017624976801
0.000026113052404	0.000026113052601
0.000036480135426	0.000036480144069
0.000038289865234	0.000038289857965

Table 15: Comparación de eigenvalores

Norma del residuo	
n	$\|Ax - \lambda x\|$
1	2.793×10^{-10}
2	1.050×10^{-9}
3	2.833×10^{-9}
4	3.434×10^{-9}
5	3.247×10^{-9}
6	4.807×10^{-9}
7	5.686×10^{-9}

Table 16: Comparación de la norma del residuo $\|Ax - \lambda x\|$

□

Problem 3: Desarrollar un programa que calcule los m valores propios y vectores propios más grandes para matrices de tamaño n mediante el **método de iteración de subespacio. (2 puntos)**

Solution.

```
make run-p3 ARGS="t5a/Eigen_3x3.txt t5a/phi0_3.txt 1e-12 10000 c_eigenvalues.txt
              c_eigenvectors.txt"
```

```
python3 p1_verify.py t5a/Eigen_3x3.txt 3 0
```

Eigenvectores	
Numpy	Iter. de subespacio
$[-0.175\ \ 0.644\ \ -0.745]$	$[0.175\ \ -0.644\ \ 0.745]$
$[0.365\ \ -0.660\ \ -0.657]$	$[0.365\ \ -0.660\ \ -0.657]$
$[-0.914\ \ -0.387\ \ -0.120]$	$[-0.914\ \ -0.387\ \ -0.120]$

Table 17: Comparación de resultados de Eigen_3x3.txt

Eigenvalores	
Numpy	Iter. de subespacio
11.538405	11.538405
8.213809	8.213809
4.247786	4.247786

Table 18: Comparación de eigenvalores de Eigen_3x3.txt

```
make run-p3 ARGS="t5a/Eigen_5x5.txt t5a/phi0_5.txt 1e-12 10000 c_eigenvalues.txt
              c_eigenvectors.txt"
```

```
python3 p1_verify.py t5a/Eigen_5x5.txt 3 1
```

Eigenvalores	
Numpy	Iter. de subsespacio
700.030781317611741	700.030781317611854
60.028366199091948	60.028366199091877
8.338447114875077	8.338447114875077

Table 19: Comparación de eigenvalores

Eigenvectores	
Numpy	Iter. de subespacio
$[-0.000\ \ 0.001\ \ 0.003\ \ -0.006\ \ -0.999]$	$[-0.000\ \ 0.001\ \ 0.003\ \ -0.006\ \ -0.999]$
$[-0.017\ \ 0.018\ \ -1.000\ \ -0.000\ \ -0.003]$	$[0.017\ \ -0.018\ \ 1.000\ \ 0.000\ \ 0.003]$
$[0.122\ \ 0.226\ \ 0.002\ \ 0.966\ \ -0.005]$	$[-0.122\ \ -0.226\ \ -0.002\ \ -0.966\ \ 0.005]$

Table 20: Comparación de resultados de Eigen_5x5.txt

```
make run-p3 ARGS="t5a/Eigen_50x50.txt t5a/phi0_50.txt 1e-12 10000
              c_eigenvalues.txt c_eigenvectors.txt"
```

Eigenvalores	
Numpy	Iter. de subespacio
500.001542724573085	500.001542724574108
490.001431174117840	490.001431174118466
480.000526650997813	480.000526650998040
470.001058080440032	470.001058080439805
460.001269023366604	460.001269023366888
450.000483273630266	450.000483273628618
440.000418898496378	440.000418898496832

Table 21: Comparación de eigenvalores 50x50

Norma del residuo	
n	$\|Ax - \lambda x\|$
1	6.9997×10^{-7}
2	5.5789×10^{-9}
3	1.5060×10^{-9}
4	1.2245×10^{-9}
5	1.0248×10^{-9}
6	8.2062×10^{-10}
7	3.7819×10^{-10}

Table 22: Comparación de la norma del residuo $\|Ax - \lambda x\|$ 50x50

```
make run-p3 ARGS="t5a/Eigen_125x125.txt t5a/phi0_125.txt 1e-6 10000
              c_eigenvalues.txt c_eigenvectors.txt"
```

Eigenvalores	
Numpy	Iter. de subespacio
1.000000000000000	0.999999999999999
1.000000000000000	0.999999999999999
1.000000000000000	0.999999999999999
1.000000000000000	0.999999999999933
1.000000000000000	0.999999999992407
1.000000000000000	0.999999999990629
1.000000000000000	0.999999999990306

Table 23: Comparación de eigenvalores 125x125

Norma del residuo	
n	$\|Ax - \lambda x\|$
1	1.385×10^{-9}
2	1.370×10^{-9}
3	1.321×10^{-9}
4	2.584×10^{-10}
5	2.755×10^{-9}
6	3.061×10^{-9}
7	3.113×10^{-9}

Table 24: Comparación de la norma del residuo $\|Ax - \lambda x\|$ 125x125

□

Problem 4: Desarrollar un programa que calcule los m valores propios y vectores propios más pequeños para matrices de tamaño n mediante el **método de iteración de subespacio. (2 puntos)**

Solution.

```
make run-p4 ARGS="t5a/Eigen_3x3.txt t5a/phi0_3.txt 1e-12 10000 c_eigenvalues.txt
              c_eigenvectors.txt"
```

Eigenvalores	
Numpy	Iter. de subespacio
4.247785692735033	4.247785692735035
8.213809393221592	8.213809393221592
11.538404914043371	11.538404914043371

Table 25: Comparación de eigenvalores de Eigen_3x3.txt

Eigenvectores	
Numpy	Iter. de subespacio
$[-0.914\ \ -0.387\ \ -0.120]$	$[-0.914\ \ -0.387\ \ -0.120]$
$[0.365\ \ -0.660\ \ -0.657]$	$[0.365\ \ -0.660\ \ -0.657]$
$[0.175\ \ -0.644\ \ 0.745]$	$[-0.175\ \ 0.644\ \ -0.745]$

Table 26: Comparación de resultados de Eigen_3x3.txt

```
make run-p4 ARGS="t5a/Eigen_5x5.txt t5a/phi0_5.txt 1e-12 10000 c_eigenvalues.txt
              c_eigenvectors.txt"
```

Eigenvalores	
Numpy	Iter. de subespacio
0.057074741004227	0.057074741004193
3.745330627417469	3.745330627417476
8.338447114875077	8.338447114875084

Table 27: Comparación de eigenvalores 5x5

Eigenvectores	
Numpy	M. de Potencia
$[-0.992\ \ -0.003\ \ 0.016\ \ 0.126\ \ -0.001]$	$[-0.992\ \ -0.003\ \ 0.016\ \ 0.126\ \ -0.001]$
$[0.031\ \ -0.974\ \ -0.018\ \ 0.224\ \ -0.003]$	$[0.031\ \ -0.974\ \ -0.018\ \ 0.224\ \ -0.003]$
$[-0.122\ \ -0.226\ \ -0.002\ \ -0.966\ \ 0.005]$	$[-0.122\ \ -0.226\ \ -0.002\ \ -0.966\ \ 0.005]$

Table 28: Comparación de resultados de Eigen_5x5.txt

```
make run-p4 ARGS="t5a/Eigen_50x50.txt t5a/phi0_50.txt 1e-12 10000
              c_eigenvalues.txt c_eigenvectors.txt"
```

Eigenvalores	
Numpy	M. de Potencia
9.998050174885066	9.998050174885130
19.998793857960756	19.998793857960724
29.998930953123583	29.998930953123381
39.999763292782198	39.999763292782021
49.998366378355584	49.998366378355612
59.999844677982182	59.999844677982381
69.999135451279130	69.999135451279230

Table 29: Comparación de eigenvalores 50x50

Norma del residuo	
n	$\|Ax - \lambda x\|$
1	5.817×10^{-9}
2	6.999×10^{-7}
3	6.577×10^{-9}
4	1.845×10^{-9}
5	1.944×10^{-10}
6	3.323×10^{-10}
7	9.247×10^{-10}

Table 30: Comparación de la norma del residuo $\|Ax - \lambda x\|$ 50x50

```
make run-p4 ARGS="t5a/Eigen_125x125.txt t5a/phi0_125.txt 1e-6 10000
              c_eigenvalues.txt c_eigenvectors.txt"
```

Eigenvalores	
Numpy	Iter. de subespacio
0.000001862755663	0.000001948721578
0.000008134856129	0.000008260593383
0.000016501222866	0.000016891501668
0.000017624976804	0.000017509493655
0.000026113052404	0.000026125225977
0.000036480135426	0.000037249559788
0.000038289865234	0.000037728586215

Table 31: Comparación de eigenvalores 125x125

Norma del residuo	
n	$\|Ax - \lambda x\|$
1	4.054×10^{-6}
2	4.475×10^{-6}
3	4.864×10^{-6}
4	4.593×10^{-6}
5	4.853×10^{-6}
6	4.678×10^{-6}
7	4.369×10^{-6}

Table 32: Comparación de la norma del residuo $\|Ax - \lambda x\|$ 125x125

□

Problem 5: Desarrollar un programa que resuelva un sistema de ecuaciones de la forma $\mathbf{Ax} = \mathbf{b}$ utilizando el **método de Gradiente Conjugado. (2 puntos)**

Solution.
Usando `make run-p5 ARGS="t5a/A1.txt t5a/b1.txt t5a/x1.txt"` obtenemos el vector

$$[3.012\ \ -2.409\ \ 7.128]$$

Con python, verificamos con `python3 p5.py t5a/A1.txt t5a/b1.txt`, que nos regresa el mismo vector:

$$[3.01157697\ \ -2.40863579\ \ 7.12797247]$$

Ahora, con la matriz grande, usando `make run-p5 ARGS="t5a/A2.txt t5a/b2.txt t5a/x2.txt"`, y comparando con

```
python3 p5.py t5a/A2.txt t5a/b2.txt
```

tenemos una diferencia de 0.00124.

□