The power method is used to calculate the largest eigenvalue (in absolute value) of a matrix as well as the associated eigenvector.

Let E be a vector space and f an endomorphism of E (a linear application of E in E). We say that λ is an eigenvalue of f if and only if:

$$\lambda \in \mathbb{R} \text{ or } \mathbb{C}, \exists x \in E, x \neq 0$$
$$f(x) = \lambda x$$

Let $A = \left(a_{ij}\right)$ be the matrix of f. We can then write:

$$Ax = \lambda x$$

Power method:

Let A be a square matrix of order n and let $(\lambda_1, \lambda_2, \ldots, \lambda_n)$ be its eigenvalues such that:

$$|\lambda_1| < |\lambda_2| < \ldots < |\lambda_n|$$

Let $u_1, u_2, \ldots, u_n$ be the associated eigenvectors. Any vector x_0 can be written as follows in the eigenvector basis:

$$x_0 = a_1 u_1 + a_2 u_2 + \cdots + a_n u_n$$

Let's assume that $a_n \neq 0$, we calculate the sequence x_{k+1}=Ax_k:

$$\begin{aligned} x_k &= Ax_{k-1} = A\left(Ax_{k-2}\right) = \ldots = A^k x_0 \\ &= A^k\left(a_1 u_1 + a_2 u_2 + \cdots + a_n u_n\right) \\ &= \lambda_1^k a_1 u_1 + \lambda_2^k a_2 u_2 + \cdots + \lambda_n^k a_n u_n \\ &= \lambda_n^k \left(\left(\frac{\lambda_1}{\lambda_n}\right)^k a_1 u_1 + \left(\frac{\lambda_2}{\lambda_n}\right)^k a_2 u_2 + \cdots + a_n u_n\right) \end{aligned}$$

if $i \neq n$, then $\left(\frac{\lambda_i}{\lambda_n}\right)^k \rightarrow 0$ when k becomes large and thus the dominant term becomes $\lambda_n^k a_n u_n$

After a certain number of iterations k we can deduce:

- The largest eigenvalue $|\lambda_n| \sim \frac{\|x_{k+1}\|}{\|x_k\|}$
- The associated eigenvector: $u_n \sim x_k$

The method remains valid if λ_n is multiple. On the other hand, if we have $\lambda_n \neq \lambda_{n-1}$ and $|\lambda_n| = |\lambda_{n-1}|$, the demonstration is not valid. This is what happens in particular in the case of complex eigenvalues.

Below, my (basic) implementation of the power method in Python. I then calculate the largest eigenvalue and the associated eigenvector of the matrix:

$$\begin{pmatrix} 5 & 1 & 2 & 0 & 4 \\ 1 & 4 & 2 & 1 & 3 \\ 2 & 2 & 5 & 4 & 0 \\ 0 & 1 & 4 & 1 & 3 \\ 4 & 3 & 0 & 3 & 4 \end{pmatrix}$$

In [1]:
```
#Abdelwahid Benslimane

import numpy as np
from math import *

def powerMethodAlgo(A, x) :

    # x is a vector used to initiate the algorithm
    #A is the input matrix of which we want to calculate the largest eigenvalue and the associated eigenvector

    vOld = x.copy()/np.linalg.norm(x,2) #vOld is the eigenvector calcultaed at iteration i. At the beginning vOld is the
                                        #vector x divided by its L2 norm. The normalization is here avoid
                                        #exceeding the capacity due to too large values
    vNew = A.dot(vOld.copy()) #vNew is the eigenvector calculated at iteration i+1. At the beginning VNew is the product
                              #of the input matrix A and vOld
    eigValOld = float('nan') #eigValOld is the eigenvalue calculated at iteration i
    eigValNew = vOld.copy().T.dot(vNew.copy()) #eigValNew is the eigenvalue calculated at iteration i+1. At the beginning
                                               #it is the product of cOld and vNew

    for i in range(100000): # 100000 is the maximum number of iterations arbitrarily chosen



        if np.isclose(eigValOld, eigValNew): #the shutoff parameter of the algorithm is eigValNew and eighValOld
                                             #to be close enough to each other, which would mean that the algorithm
                                             #has converged to the largest eigenvalue of the matrix
                                             #we return the largest eigenvalue eighValNew and the associated
                                             #eigenvector vOld

            return eigValNew, vOld

        else:

            #if we enter a new iteration, eigValOld is replaced by eigValNew, vOld is replaced by vNew divided by
```

```python
            #its L2 norm, vNew is replaced by the product of A and the (new) vOld, and eigValNew is replaced by the
            #product of the (new) vOld and the (new) vNew

            eigValOld = eigValNew.copy()
            vOld = vNew.copy()/ np.linalg.norm(vNew.copy(),2)
            vNew = A.dot(vOld)
            eigValNew = vOld.T.dot(vNew)

    #if we didn't converge to the largest eigenvalue we return 0
    return  0, 0
```

In [2]:
```python
A = np.array([ [5., 1., 2., 0., 4.], [1., 4., 2., 1., 3.], [2., 2., 5., 4., 0.],
               [0., 1., 4., 1., 3.], [4., 3., 0., 3., 4.]])

N = np.shape(A)[0] #N = number of lines of the input matrix
randX = np.random.rand(N) #randX is a random vector of which the length equals the number of lines
                          #(or number of columns) of the matrix A, it is the vector used
                          #to initiate the power method

eigvMax, eigvec = powerMethodAlgo(A, randX)

print("the largest eigenvalue of the matrix A is:")
print(eigvMax)
print("the associated eigenvector is:")
print(eigvec)
```

```
the largest eigenvalue of the matrix A is:
12.025726139573655
the associated eigenvector is:
[0.48418871 0.41137698 0.45216883 0.34404886 0.5229761 ]
```

In [ ]: