Deflation method:

Let A be a diagonalizable matrix and $u_1, u_2, \dots, u_n$ its eigenvectors basis, there exists a (dual) basis $v_1, v_2, \dots, v_n$ such that:

$\forall i, j \quad {}^t v_i u_j = \delta_{ij}$

with $\delta_{ij} = 0$ if $i \neq j$ and $\delta_{ii} = 1$

Let B be a matrix such that:

$B = A - \lambda_n u_n {}^t v_n$

with λ_n being the largest eigenvalue (in absolute value) of A (whose eigenvalues, arranged in ascending order of the absolute values, are $\lambda_1, \lambda_2, \dots, \lambda_n$)

$$\begin{aligned}&\forall i \in \{1, 2, \dots, n\}\\ &Bu_i = Au_i - \lambda_n u_n ({}^t v_n u_i)\\ &= \lambda_i u_i - \lambda_n u_n \delta_{ni}\\ &= \begin{cases} 0 \text{ if } i = n \\ \lambda_i u_i \text{ if } i \neq n \end{cases}\end{aligned}$$

The eigenvalues of B are therefore $\lambda_1, \lambda_2, \dots, \lambda_{n-1}$ and 0 and its eigenvectors are the same eigenvectors as A.

If we apply the power method to B, we obtain the eigenvalue λ_{n-1} and the associated eigenvector u_{n-1}.

We must now define the $v_1, v_2, \dots, v_n$ basis thanks to the knowledge of the eigenvalue λ_n and the associated eigenvector u_n obtained by applying the power method to the matrix A.

Let's consider the matrix ${}^t A$ which is the transpose of the matrix A. ${}^t A$ has the same eigenvalues as A. Let v be the eigenvector of ${}^t A$ associated to the eigenvalue λ_n.

$$\begin{aligned}{}^t v A u_i &= \left({}^t v A\right) u_i = \lambda_n {}^t v u_i\\ &= {}^t v \left(A u_i\right) = \lambda_i {}^t v u_i\end{aligned}$$

Therefore, if $i \neq n, {}^t v u_i = 0$. We can conclude that $v_1, v_2, \dots, v_n$ is the eigenvectors basis of ${}^t A$.

The deflation method combined with the power method allows to find all the eigenvalues of a diagonalizable matrix and the associated eigenvectors. However, as the method is iterated, there is an accumulation of error as the power method approximates the largest eigenvalue and the associated eigenvector, the matrix B is therefore not exactly right.

Below, my (basic) implementation of the deflation method in Python as well as my implementation of the power method that I already explained. I then apply the deflation method to 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]:
```python
#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
            #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]:

```
def deflation(A):


    A = A.copy() #we copy the input matrix
    #we assert that the matrix is squared and we print a message if it is not the case
    assert np.shape(A)[0] == np.shape(A)[1], 'the input matrix must be a square matrix'
    N = np.shape(A)[0] #N = number of lines of the input matrix

    eigValv = np.zeros(N) # we create a vector where the eigenvalues will be stored
                           #it contains zeros at the biginning
    eigVecta = np.zeros([N, N]) #we create an array where eigenvectors will be stored. It contains zeros at the beginning
    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
    solution = np.zeros(2)    #soution is a vector that will store the output of the powerMethodAlgo function

    #we calculate the N eigenvalues and eigenvectors of A
    for i in range(N):
        solution = powerMethodAlgo(A, randX)
        eigValv[i] = solution[0]
        eigVecta[i] = solution[1]

        A = A - eigValv[i]*np.outer(eigVecta[i], eigVecta[i]).T/np.linalg.norm(eigVecta[i], 2)

        #we return the eigenvalues and the eigenvectors, please note that the eigenvectors must be read vertically,
        #not horizontally, as eigVecta is transposed before being returned

    return eigValv, eigVecta.T
```

In [3]:

```
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.]])

a,b = deflation(A)
print("eigenvalues:")
print(a)
print("eigenvectors:")
print(b)
```

```
eigenvalues:
[12.02574566  5.67253861 -3.55760236  3.36217497  1.4976621 ]
eigenvectors:
[[ 0.48452673 -0.44176628  0.30839565 -0.6142211  -0.30540104]
 [ 0.41133594  0.01046019  0.18597203  0.69071651 -0.56828059]
 [ 0.45150641  0.7118287  -0.42937859 -0.29659096 -0.15154447]
 [ 0.34340298  0.33231471  0.66148218  0.08796996  0.56983461]
 [ 0.52369165 -0.43312813 -0.49837925  0.22346865  0.48591169]]
```

In [ ]: