Deflation Method with Inverse Power Iteration for Computing the Smallest Eigenvalues of Symmetric Matrices: Analysis and Implementation

June 2023

Nicola Orecchini
Politecnico di Torino
s303998@studenti.polito.it

Leonardo Moraglia
Politecnico di Torino
s309345@studenti.polito.it

Contents

I. Introduction

This report constitutes the "Free Homework" for the final exam of the course "Computational Linear Algebra for Large Scale Problems", held during the academic year 2022-2023. In this homework, we will implement, in the programming language Python, the Deflation Method with Inverse Power Iteration to compute the M smallest eigenvalues of a symmetric matrix.

In particular, the question we aim to empirically ask within our study is whether the Deflation Method combined with the Inverse Power Method can be used for the computation of some small eigenvalues of a symmetric matrix, and, more specifically, how many of them can be considered a good approximation, before the introduced bias takes over, making the method unstable and the results unreliable.

The report is subdivided into sections, providing a complete overview of the work done: in II we present the techniques used for the approximation of the M smallest eigenvalues of a symmetric matrix (i.e., Deflation Method and Inverse Power Method); in III we explain how we implemented them in Python, and in IV we compare our results with those obtained using some already built methods. Finally, in V we summarize our findings.

II. Analysis

We start from the problem of computing some of the smallest eigenvalues of a symmetric matrix. So, among the numerical methods we studied, we concentrate on those designed specifically to compute some eigenvalues (contrary to other methods used to compute all the eigenvalues).

1) Deflation Method

In particular, we start by taking in consideration the Deflation Method for symmetric matrices.

Let A be a real symmetric matrix in $\mathcal{R}^{nxn}$. Then, it can be proved that all its eigenvalues λ_i are real, and the associated eigenvectors $\mathbf{x_i}$ are orthogonal and orthonormal. Furthermore, A is diagonalizable, i.e. we can write

$$A = X\Lambda X^T = \begin{bmatrix} \left(\mathbf{x_1}\right) & \left(\mathbf{x_2}\right) & \dots & \left(\mathbf{x_n}\right) \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ \vdots & & & \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} \begin{bmatrix} (& \mathbf{x_1} &) \\ (& \mathbf{x_2} &) \\ & \vdots & \\ (& \mathbf{x_n} &) \end{bmatrix}$$

where $\mathbf{x_i}$ are the eigenvectors. Then, take any vector $\mathbf{y} \in \mathcal{R}^n$ to have:

$$A\mathbf{y} = X\Lambda \begin{bmatrix} (& \mathbf{x_1} &) \\ (& \mathbf{x_2} &) \\ & \vdots & \\ (& \mathbf{x_n} &) \end{bmatrix} \mathbf{y}$$

that is, denoting by $\mathbf{y_E}$ the vector $\mathbf{y}$ written with respect to the canonical basis, whereas $\mathbf{y_X}$ the same vector but written with

respect to the basis given by X,

$$
\begin{aligned}
A\mathbf{y_E} &= X\Lambda\mathbf{y_X} = \\
&= X\Lambda \begin{bmatrix} \mathbf{x_1}^T\mathbf{y} \\ \vdots \\ \mathbf{x_n}^T\mathbf{y} \end{bmatrix} \\
&= \begin{bmatrix} \left(\mathbf{x_1}\right)\lambda_1 & \left(\mathbf{x_2}\right)\lambda_2 & \dots & \left(\mathbf{x_n}\right)\lambda_n \end{bmatrix} \begin{bmatrix} \mathbf{x_1}^T\mathbf{y_E} \\ \vdots \\ \mathbf{x_n}^T\mathbf{y_E} \end{bmatrix} \\
&= \sum_{i=1}^{n} \left(\mathbf{x_i}\right)\lambda_i(\mathbf{x_i^T})\mathbf{y_E}
\end{aligned}
$$

Now, given that $\mathbf{y_E}$ does not depend on i, we can bring it out of the summation, and also we can bring the constant λ_i at the front of the expression, and thus obtaining the relation

$$A = \sum_{i=1}^{n} \lambda_i \left(\mathbf{x_i}\right)(\mathbf{x_i^T})$$

In the end, we wrote the matrix A (that is by hypothesis a symmetric arbitrary matrix) as a linear combination of matrices obtained as $\mathbf{x_i x_i^T}$, that, by construction, all have rank 1. As the next step, we note that we could write the obtained relation by isolating the first term (that we can conventionally assume corresponds to the largest eigenvalue of A):

$$A = \lambda_1 \mathbf{x_1 x_1^T} + \sum_{i=2}^{n} \lambda_i \mathbf{x_i x_i^T}$$

and, if we define

$$A_1 := \sum_{i=2}^{n} \lambda_i \mathbf{x_i x_i^T}$$

we have the following:

Claim: If A has

eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$(with $|\lambda_1| > |\lambda_2| > \dots > |\lambda_n|$)
eigenvectors $x_1, x_2, \dots, x_n$

then, the matrix A_1 obtained as above has

eigenvalues $0, \lambda_2, \dots, \lambda_n$(with $|\lambda_2| > \dots > |\lambda_n|) > 0$
eigenvectors $x_1, x_2, \dots, x_n$

i.e., A_1 has the same eigenvalues except λ_1, which is replaced by 0, and the same eigenvectors.

Proof: Let's look for the eigenvalues and eigenvectors of A_1. First, we proof that 0 is an eigenvalue with eigenvector x_1:

$$
\begin{aligned}
A_1\mathbf{x_1} &= (A - \lambda_1 \mathbf{x_1 x_1^T})\mathbf{x_1} \\
&= A\mathbf{x_1} - \lambda_1 \mathbf{x_1 x_1^T}\mathbf{x_1} \\
&\left[\mathbf{x_1^T x_1} = 1 \text{ by orthonormality of the basis}\right] \\
&= A\mathbf{x_1} - \lambda_1 \mathbf{x_1} \\
&\left[\mathbf{x_1} \text{ is eigenvector of A, with } \lambda_1 \text{ eigenvalue}\right] \\
&= \lambda_1 \mathbf{x_1} - \lambda_1 \mathbf{x_1} = 0 \\
&\Rightarrow A_1\mathbf{x_1} = 0\mathbf{x_1} \\
&\Rightarrow 0 \text{ is eigenvalue of } A_1, \\
&\text{with corresponding eigenvector } \mathbf{x_1}
\end{aligned}
$$

Then, for $i \neq 1$, we have:

$$
\begin{aligned}
&A_1\mathbf{x_i} = A\mathbf{x_i} - \lambda_1 \mathbf{x_1 x_1^T}\mathbf{x_i} = \\
&\left[\mathbf{x_p^T x_q} = 0, \forall p \neq q, \text{because of orthonormality of basis}\right] \\
&= A\mathbf{x_i}
\end{aligned}
$$

Thus, every other eigenvalue and eigenvector of A_1 is same of A.

With the just presented argument, we can describe the Deflation Method: it consists of calculating the M largest or smallest eigenvectors of a matrix A one at a time, each time by removing from A a matrix obtained by multiplying the found eigenvector with its own transposition and scaling it by the found eigenvalue. At each step of the method, we only need to compute the largest or the smallest eigenvalue of the resulting matrix. Of course, this method can quickly lead to instability, as approximation errors propagate at each step, but it can still be used to compute some of the largest or smallest eigenvalues of a symmetric matrix, which comes in handy for multiple applications, e.g. the Spectral Clustering (which is discussed in another Homework for this course).

2) Inverse power method

At this point, we can introduce the Inverse Power Method, that constitutes the "kernel" of the Deflation Method, i.e. the algorithm we use at each iteration of the Deflation Method to find the smallest eigenvalue of the matrix. Of course this is not the unique method one can use as the kernel, but we will see that the Power Method, specifically in its Inverse variation, has some nice convergence properties for symmetric matrices, which are at the core of our study.

The power method is used to compute the largest in module eigenvector (and its corresponding eigenvalue). Let's see how it works.
Let A be a real matrix in $\mathbb{R}^{n\times n}$ (not necessarily symmetric), with eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$, such that $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n| \geq 0$, and with corresponding eigenvectors $x_1, \dots, x_n$. Then, any vector $v_0 \in \mathbb{R}^n$ can be written as

$$v_0 = \alpha_1 x_1 + \dots + \alpha_n x_n$$

Now, define

$$\begin{aligned} v_1 := Av_0 &= \alpha_1 Ax_1 + \cdots + \alpha_n Ax_n \\ &= \alpha_1\lambda_1 x_1 + \cdots + \alpha_n\lambda_n x_n \\ &= \lambda_1\left[\alpha_1 x_1 + \alpha_2(\frac{\lambda_2}{\lambda_1})x_2 + \cdots + \alpha_n(\frac{\lambda_n}{\lambda_1})x_n\right] \\ &\vdots \end{aligned}$$

And so on. So, by induction

$$v_k = \lambda_1^k\left[\alpha_1 x_1 + \alpha_2(\frac{\lambda_2}{\lambda_1})^k x_n + \cdots + \alpha_n(\frac{\lambda_n}{\lambda_1})^k x_n\right], k \geq 0 \quad (1)$$

Then, we have that

$$\lim_{k\to\infty} v_k = \lambda_1^k\alpha_1 x_1, \quad (2)$$

$$\text{as } |\frac{\lambda_i}{\lambda_1}| < 1, \forall i \quad (3)$$

From this we can conclude that, after a certain number of iterations, the vector v_k (that is an arbitrary vector of $\mathbb{R}^n$) has the same direction of x_1, which is the eigenvector of A associated with the eigenvalue λ_1. In other words, we are approximating the eigenvector x_1 (which we don't have) with v_k (that is simply the application of A k times to the random vector v_0).
A problem arises in this procedure: at each iteration, we are finding v_k as

$$v_k = Av_{k-1} = A^k v_0 \approx \lambda_1^k\alpha_1 x_1$$

and, if $|\lambda_1| > 1$, then this quantity may grow indefinitely, leading to an overflow error. On the other hand, if $|\lambda_1| < 1$, then we could encounter an underflow error. We resolve this problem by normalizing at each step the found vector (and thus still maintaining its alignment to x_1):

$$v_1 \leftarrow \frac{v_1}{||v_1||}$$

Still, we are missing the eigenvalue: how to find λ_1?
We have multiple ways available to calculate it: a computationally cheap option is the following. Starting from Equation (2), we can write

$$\lim_{k\to\infty} \frac{1}{\lambda_1^k} v_k = \alpha_1 x_1,$$

that is a vector relation, i.e. contains n limits, one for each component of the vector. So, if we fix a component i, and consider the ratio of the limits at iterations $k+1$ and k, we have

$$\lim_{k\to\infty} \frac{\frac{1}{\lambda_1^{k+1}} v_{k+1}[i]}{\frac{1}{\lambda_1^k} v_k[i]} = \frac{\alpha_1 x_1[i]}{\alpha_1 x_1[i]} = 1.$$

And thus

$$\lim_{k\to\infty} \frac{v_{k+1}[i]}{v_k[i]} = \lambda_1. \quad (4)$$

In other words, we could take the ratio between the component i of the v vector in two subsequent iterations, and eventually get the eigenvalue. This procedure is very cheap, but requires a strong assumption, i.e. that every component is different from zero.
This is the reason why other methods are available: to avoid such an assumption. Of course different methods could come with some computational cost, but here we report the case of the Rayleigh Quotient, which has nice convergence properties for symmetric matrices.
The Rayleigh Quotient is defined as follows:

$$R_a^k := \frac{v_k^T A v_k}{v_k^T v_k} = \frac{v_k^T v_{k+1}}{v_k^T v_k}$$

We immediately note how this ratio, with respect to that of Equation (4), involves computing a Euclidean norm at each step, making it more computational costly instead of just taking an element in the vector. Anyway, we will see what advantages this ratio offers us for our specific case.

First, we show how this ratio tends to λ_1. From 1 we have

$$\begin{aligned} v_k &= \lambda_1^k\left[\alpha_1 x_1 + \alpha_2(\frac{\lambda_2}{\lambda_1})^k x_n + \cdots + \alpha_n(\frac{\lambda_n}{\lambda_1})^k x_n\right] = \\ &:= \lambda_1^k u_k \end{aligned}$$

and, equivalently,

$$v_{k+1} = \lambda_1^{k+1} u_{k+1}$$

We can use these two quantities to calculate numerator and denominator of the Rayleigh Quotient. Note that we highlight in blue the terms containing a multiplication of the vector x_i by itself, and in orange those containing a multiplication between an x_i and an x_j, with $i \neq j$. This will be useful later, when we will analyse the application of this method to symmetric matrices (note that what we said about the Power Method until now does not suppose A symmetric). Denominator:

$$\begin{aligned} v_k^T v_k &= (\lambda_1^k u_k)^T(\lambda_1^k u_k) = \lambda_1^{2k} u_k^T u_k \\ &= \lambda_1^{2k}\Big[\alpha_1^2 \boxed{x_1^T x_1} + \\ &\alpha_1\alpha_2(\frac{\lambda_2}{\lambda_1})^k \boxed{x_1^T x_2} + \\ &\alpha_1\alpha_3(\frac{\lambda_3}{\lambda_1})^k \boxed{x_2^T x_3} + \\ &\alpha_2\alpha_1(\frac{\lambda_2}{\lambda_1})^k \boxed{x_2^T x_1} + \\ &\alpha_2^2(\frac{\lambda_2}{\lambda_1})^{2k} \boxed{x_2^T x_2} + \ldots\Big] \end{aligned}$$

Numerator:

$$\begin{aligned} v_k^T v_{k+1} &= (\lambda_1^k u_k)^T(\lambda_1^{k+1} u_{k+1}) = \lambda_1^{2k+1} u_k^T u_{k+1} \\ &= \lambda_1^{2k+1}\Big[\alpha_1^2 \boxed{x_1^T x_1} + \\ &\alpha_1\alpha_2(\frac{\lambda_2}{\lambda_1})^{k+1} \boxed{x_1^T x_2} + \\ &\alpha_2\alpha_1(\frac{\lambda_2}{\lambda_1})^k \boxed{x_2^T x_1} + \\ &\alpha_2^2(\frac{\lambda_2}{\lambda_1})^{2k+1} \boxed{x_2^T x_2} + \ldots\Big] \end{aligned}$$

Then, we can evaluate the following

$$\lim_{k\to\infty} \frac{v_k^T v_{k+1}}{v_k^T v_k} = \frac{\lambda_1^{2k+1}}{\lambda_1^{2k}} \frac{\left[\alpha_1^2 x_1^T x_1 + O[(\frac{\lambda_2}{\lambda_1})^{k+1}] + O[(\frac{\lambda_2}{\lambda_1})^{2k+1}]\right]}{\left[\alpha_1^2 x_1^T x_1 + O[(\frac{\lambda_2}{\lambda_1})^{k}] + O[(\frac{\lambda_2}{\lambda_1})^{2k}]\right]} =$$

$$= \lambda_1 \frac{\alpha_1^2 x_1^T x_1}{\alpha_1^2 x_1^T x_1} = \lambda_1$$

So, we showed that the Rayleigh Quotient tends to the largest eigenvalue of A, λ_1. Let's consider the rate of convergence of this series of approximations:

$$\text{eigenvalues: } |\lambda_1 - \lambda_1^{(k)}| \leq C(\frac{\lambda_2}{\lambda_1})^k, \text{with C} \in \mathbb{R}$$

$$\text{eigenvectors: } ||x_1 - v_k|| \leq D(\frac{\lambda_2}{\lambda_1})^k, \text{with D} \in \mathbb{R}$$

So, the rate of convergence in given by a power of the ratio between the second largest eigenvalue and the largest one.
Let's now introduce the assumption of A symmetric matrix. A property of real symmetric matrices is that an orthonormal basis of eigenvectors exists, where the scalar product bewteen two different eigenvectors is zero, while the scalar of ae eigenvector with itself is 1. So, if we assume A real symmetric matrix, the blue terms in the calculations of the numerator and denominator of the Rayleigh Quotient are all 1, whereas the orange ones are 0.
This allows us only to take into account terms involving $(\frac{\lambda_2}{\lambda_1})^{2k}$ and $(\frac{\lambda_2}{\lambda_1})^{2k+1}$, eventually leading to the following result about convergence of eigenvalues and eigenvectors:

$$\text{eigenvalues: } |\lambda_1 - \lambda_1^{(k)}| \leq C(\frac{\lambda_2}{\lambda_1})^{2k}, \text{with C} \in \mathbb{R}$$

$$\text{eigenvectors: } ||x_1 - v_k|| \leq D(\frac{\lambda_2}{\lambda_1})^{2k}, \text{with D} \in \mathbb{R}$$

In other words, the convergence rate of the Power Method is doubled, if A is symmetric. This is what drives our choice of implementing it to work in synergy with the Deflation Method to find the smallest eigenvalues of a symmetric matrix.

One last step is required before moving on to the implementation: the Power Method works for finding the largest eigenvalue of a matrix, whereas our problem is that of finding the smallest. Here is why whe introduce the Inverse Power Method.

We can slightly modify the Power Method to compute the smallest eigenvalue: assume matrix A has eigenvalues $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$. If x_i is an eigenvector of A, then

$$Ax_i = \lambda_i x_i$$

$$\left[\text{A symmetric } \implies \exists A^{-1}\right]$$

$$A^{-1}Ax_i = \lambda_i A^{-1} x_i$$

$$\left[\text{A symmetric } \implies \exists \text{ orthonormal basis}\right]$$

$$x_i = \lambda_i A^{-1} x_i \iff A^{-1}x_i = \frac{1}{\lambda_i} x_i$$

Then, the eigenvalues of A^{-1} are the inverse of those of A, and they satisfy $\frac{1}{|\lambda_n|} > \frac{1}{|\lambda_{n-1}|} \geq \ldots \geq \frac{1}{|\lambda_1|}$.
Hence, if we apply the power method to A^{-1}, the algorithm will find $1/\lambda_n$, which corresponds to the smallest eigenvalue of matrix A.

In the practical implementation, instead of computing A^{-1} directly (which could be very expensive), we first compute the LU factorization of matrix A. Then, at each step, to find the eigenvector $x^{(k+1)}$ we solve the equivalent equation $Ax^{(k+1)} = x^{(k)}$, instead of having to solve $x^{(k+1)} = A^{-1}x^{(k)}$.

Now that all the theoretical background has been given, we move on to the implementation.

III. Implementation

We present here the two key functions we implemented: **inverse_power_method** (Algorithm 1) and **deflation_method** (Algorithm 2). In particular, we will give an idea of the tools we used for the implementation in Python and their structure.

The **inverse_power_method** function starts by initializing an initial vector in a random way, and then proceeds with a series of iterations. In each iteration the current vector is normalized, and then the matrix equation $Ax = \lambda x$ is solved using the LU factorization (which is calculated a priori, outside of the for loop). Note that this equation is equivalent to finding the eigenvector x associated with the smallest eigenvalue λ. The result is then normalized again, and the process is repeated until convergence is achieved or the maximum number of iterations is reached. The final outcome of the "inverse_power_method" function is the approximation of the smallest eigenvalue and its corresponding eigenvector.

The **deflation_method** function, as we have seen, allows the "inverse_power_method" (which is designed to only find the smallest eigenvalue) to iteratively find multiple eigenvalues. It is designed to compute multiple smallest eigenvalues by iteratively deflating the matrix after each eigenvalue is computed. The function takes as input the symmetric matrix A and the desired number of eigenvalues, M. Within the function, the "inverse_power_method" is invoked M times, each time computing a single smallest eigenvalue. After an eigenvalue and its eigenvector are obtained, the matrix is deflated by subtracting the outer product of the eigenvector from itself, effectively removing the contribution of the computed eigenvalue from the matrix. This deflation process reduces the matrix's dimensionality and allows for the calculation of subsequent eigenvalues. The "deflation_method" continues this process until M eigenvalues are computed and their corresponding eigenvectors are obtained.

Algorithm 1 Inverse Power Method for Computing Smallest Eigenvalue and Eigenvector

function INVERSE_POWER_METHOD($A, niterations\_max = 50, tol = 1e-15$)
  Input:
    A: Sparse matrix
    $niterations\_max$: Maximum number of iterations (default: 50)
    tol: Tolerance for convergence (default: 1e-15)
  Output:
    $smallest\_eigenvalue$: Computed smallest eigenvalue
    $eigenvector$: Normalized eigenvector corresponding to the smallest eigenvalue

  $xn \leftarrow$ Create a matrix of zeros with dimensions $(A.shape[0], niterations\_max + 1)$
  $xn[:, 0] \leftarrow$ Initialize with a random vector of ones with noise
  $v\_old \leftarrow xn[:, 0]/\|xn\|$
  $lambda\_1\_inv\_new \leftarrow 1$
  $LU \leftarrow$ Perform LU factorization of A
  for $k \leftarrow 1$ to $niterations\_max$ **do**
    $lambda\_1\_inv\_old \leftarrow lambda\_1\_inv\_new$
    $v\_new \leftarrow$ Solve the linear system using LU fact.
    $lambda\_1\_inv\_new \leftarrow v\_old^T \cdot v\_new$
    $v\_old \leftarrow v\_new/\|v\_new\|$
    if $|lambda\_1\_inv\_new - lambda\_1\_inv\_old| < tol$ **then**
      break
    end if
  end for
  return $(1/lambda\_1\_inv\_new, v\_old)$
end function

Algorithm 2 Deflation Method for Computing M Smallest Eigenvalues and Eigenvectors

function DEFLATION_METHOD($A, M, niterations\_max = 50, tol = 1e-15$)
  Input:
    A: Sparse matrix
    M: Number of smallest eigenvalues to compute
    $niterations\_max$: Maximum number of iterations for inverse power method (default: 50)
    tol: Tolerance for convergence of inverse power method (default: 1e-15)
  Output:
    $eigenvalues$: Array of the M smallest eigenvalues
    $eigenvectors$: Array of the M corresponding eigenvectors

  $eigenvalues \leftarrow$ Create an array of zeros with size M
  $eigenvectors \leftarrow$ Create a matrix of zeros with dimensions $(A.shape[0], M)$

  for $i \leftarrow 0$ to $M - 1$ **do**
    $eigenvalue, eigenvector \leftarrow$ INVERSE_POWER_METHOD$(A, niterations_max, tol)$
    $eigenvalues[i] \leftarrow eigenvalue$
    $eigenvectors[:, i] \leftarrow eigenvector$

    Deflation step: Update A by removing the contribution of the computed eigenvector
    $A \leftarrow A - eigenvalue \cdot$ outer$(eigenvector, eigenvector)$
  end for

  return $eigenvalues, eigenvectors$
end function

IV. Algorithms assessment

In this section, we provide some visual assessment for the correctness of the methods we implemented.

We start with a visual assessment of the Inverse Power Method: in Figure 1, we note two things:

1) The value of the approximated eigenvalue converges indeed to its exact value, where that the exact values have been calculated using the *scipy.sparse.linalg.eigsh* [3] function of the *Scipy* Python library for the symmetric matrix, and using the *scipy.sparse.linalg.eigs* [4] function of the same library for the non-symmetric case. The error is of the order of 10^{-2} for both cases.
2) The convergence is indeed faster for the symmetric case: we can see how stability is almost immediately reached. If we zoom in the graph (this can be done by opening the file *inverse_power_method_test.py* and uncommenting the *plt.show()* command at the end), we can note that stability is reached around iteration 5-6. On the other hand, for the non-symmetric case we can visually assess how the stability is reached at approximately iteration 15-20. So, we can confirm the property that the Power Method presents for symmetric matrices, i.e. the doubling of convergence speed.

Next, we want to assess the Deflation Method function. We take in exam 4 matrices, all symmetric, and examine how the 10 smallest eigenvalues obtained with the Deflation Method with Inverse Power Method kernel compare to the theoretical eigenvalues. Also in this case, the theoretical eigenvalues have been calculated by means of the function [4].
During the assessment of the method, we stumbled upon a peculiar implementation issue of the deflation method: taking a look at Figure 3, we notice how the 10 smallest eigenvalues of the matrix obtained with our deflation method are all flat near 0, while the theoretical values are different.
We investigated deeply this issue, and found out that this was not an implementation error of ours, but a characteristic

instability of the deflation method: suppose our matrix

$$A = \lambda_1 x_1 x_1^T + \lambda_2 x_2 x_2^T$$

has eigenvalues

$$\lambda_1 = 10^{-8}, \lambda_2 = 10^{-3},$$

which is a very similar situation to ours, as we are calculating the smallest eigenvalues of a matrix. We calculate λ_1 with the inverse power method, obtaining $\hat{\lambda_1}$, with error

$$|\lambda_1 - \hat{\lambda_1}| \approx 10^{-4}$$

and proceed with the deflation step:

$$\begin{aligned} A_1 &= A - \hat{\lambda_1} x_1 x_1^T \\ &= \lambda_1 x_1 x_1^T + \lambda_2 x_2 x_2^T - \hat{\lambda_1} x_1 x_1^T \\ &= (\lambda_1 - \hat{\lambda_1}) x_1 x_1^T + \lambda_2 x_2 x_2^T. \end{aligned}$$

Now, A_1 has eigenvalues $(\lambda_1 - \hat{\lambda_1})$, which is $\approx 10^{-4}$, and λ_2, which is $\approx 10^{-3}$, and this is the key point where the deflation method has difficulty, because, at the next step, it will not "see" λ_2 as the smallest eigenvalue of A_1, as there is a smaller eigenvalue before it. So, the deflation method goes into a loop, finding at each step the same smallest eigenvalue of the initial matrix.

How can we address this? In [1], another value is proposed for the deflation step. We don't use it, but rather use this information to hypothesise that some minor "normalization" could help us make the method converge. In particular, we found out that rescaling the quantity that we subtract from the matrix at each step of the deflation method by a positive random number $\approx 10^3$ makes drastic improvements. So, we decided to use, as a scaling factor, the quantity

$$\frac{1}{\hat{\lambda_1}},$$

where $\hat{\lambda_1}$ is the approximation of λ_1 (smallest eigenvalue of A) coming from the first step of the deflation method. In other words, at each step we subtract from A the quantity

$$\frac{\hat{\lambda_i}}{\hat{\lambda_1}} \hat{x_i} \hat{x_i}^T.$$

instead of simply

$$\hat{\lambda_i} \hat{x_i} \hat{x_i}^T.$$

Note that this factor has experimentally been determined.

Finally, in Figure 3, we can assess that the deflation method is now converging to the right smallest eigenvalues of A.

V. Conclusion

To conclude this brief study, we summarize our work. We started by presenting the Deflation Method, and how it is useful to compute the first M largest or smallest eigenvalues (and corresponding eigenvectors) of a matrix. Then, we analysed the Inverse Power Method as the kernel for the previous method, which is a good choice especially for our problem, which was that of computing the first M eigenvalues of a symmetric matrix. In fact, we have seen how this method presents better convergence properties when applied to symmetric matrices. Then, we implemented the explained methods and performed a visual assessment of them, during which we found out a stability issue, and presented a possible way to fix it. Finally, to answer the question we started with, we can conclude that we can use the Deflation Method along with the Inverse Power Method to efficiently compute some of the smallest eigenvalues and eigenvectors of a symmetric real matrix. The number of eigenvalues we can consider reliable varies from application to application, but generally we have seen how the method is enough stable for the first 10 eigenvalues, provided that a convergence study of the method has been conducted specifically for the type of matrices where it will be used.

References

[1] Jeffrey Wong, Department of Mathematics, Duke University Math 361S Lecture notes Finding eigenvalues: The power method, [Accessed 21-May-2023]; https://services.math.duke.edu/~jtwong/math361-2019/lectures/Lec10eigenvalues.pdf.

[2] notebookNotebookcommunity Eigenvalue solving, [Accessed 21-May-2023]; https://notebook.community/qgoisnard/Exercice-update/03-Eigenvalue_problem_solving.

[3] Scipy Documentation scipy.sparse.linalg.eigsh, [Accessed 21-May-2023]; https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.eigsh.html#scipy.sparse.linalg.eigsh.

[4] Scipy Documentation scipy.sparse.linalg.eigs, [Accessed 21-May-2023]; https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.eigs.html.

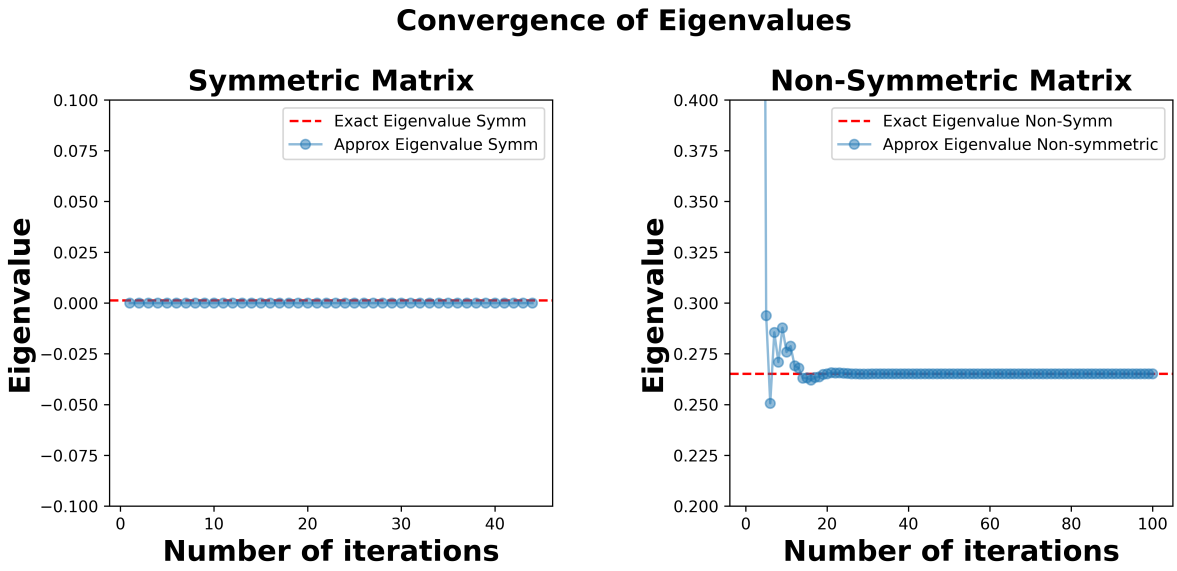

Fig. 1. Convergence of Inverse Power Method for the computation of the smallest eigenvalue of a symmetric and non-symmetric matrix, both 1000 rows x 1000 columns and with 10% sparsity.

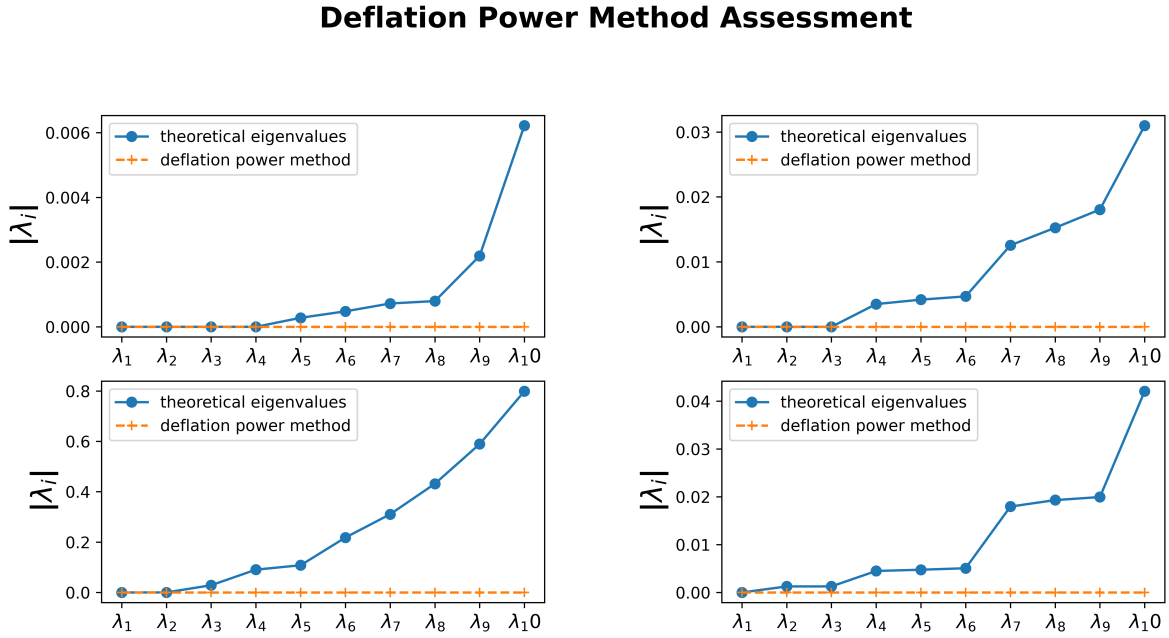

Fig. 2. Assessment of the Deflation Power Method: theoretical and estimated eigenvalues on 4 different large sparse matrices of 1000 rows x 1000 columns.

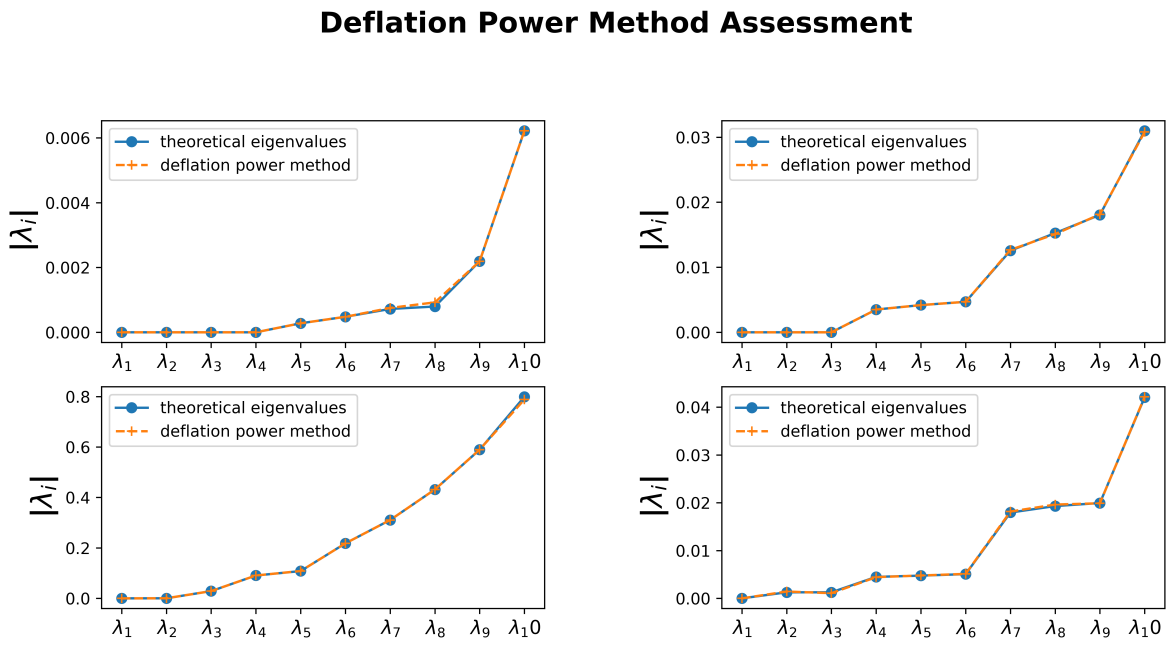

Fig. 3. Assessment of the Deflation Power Method: theoretical and estimated eigenvalues on 4 different large sparse matrices of 1000 rows x 1000 columns.