Deflation Method with Inverse Power Iteration for Computing the Smallest Eigenvalues of Symmetric Matrices

July 2023

Lorenzo Bergadano
Politecnico di Torino
lorenzo.bergadano@studenti.polito.it

Giovanni Cadau
Politecnico di Torino
giovanni.cadau@studenti.polito.it

I. Introduction

The final test for the subject ”Computational Linear Algebra for Large Scale Problems,” which was taught in the academic year 2022–2023, included this report as the ”Free Homework.” In order to get the M lowest eigenvalues of a symmetric matrix, we will develop the Deflation Method with Inverse Power Iteration in this assignment using the Python programming language.

In particular, we want to know how many small eigenvalues of a symmetric matrix can be calculated using the Deflation Method and Inverse Power Method before the introduced bias makes the method unstable and the results unreliable. This is the question that will be empirically answered in our study.

The report is organised into sections that give a thorough account of the work done: the methods for approximating the M smallest eigenvalues of a symmetric matrix, the deflation method and the inverse power method are presented in the reference section on ”Methodology” (II). Our Python implementation is described in the reference section on ”Experiment” (III); and our results are contrasted with those of other methods in the reference section on ”Results” (IV). Finally, we summarise our results in the section under ”Conclusions” (V).

II. Analysis

We begin by attempting to determine some of a symmetric matrix's lowest eigenvalues. 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 $\mathbf{v_i}$ are orthogonal and orthonormal. Furthermore, A is diagonalizable, i.e. we can write

$$A = V\Lambda V^T$$

$$= \left[\begin{pmatrix}\mathbf{v_1}\end{pmatrix} \quad \begin{pmatrix}\mathbf{v_2}\end{pmatrix} \quad \cdots \quad \begin{pmatrix}\mathbf{v_n}\end{pmatrix}\right] \begin{bmatrix}\lambda_1 & 0 & \dots & 0\\ 0 & \lambda_2 & 0 & 0\\ 0 & 0 & \lambda_3 & 0\\ \vdots & & & \\ 0 & 0 & \dots & \lambda_n\end{bmatrix} \begin{bmatrix}(& \mathbf{v_1} &)\\ (& \mathbf{v_2} &)\\ & \vdots & \\ (& \mathbf{v_n} &)\end{bmatrix}$$

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

$$A\mathbf{y} = X\Lambda \begin{bmatrix}(& \mathbf{v_1} &)\\ (& \mathbf{v_2} &)\\ & \vdots & \\ (& \mathbf{v_n} &)\end{bmatrix} \mathbf{y}$$

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

$$\begin{aligned} A\mathbf{y_E} &= V\Lambda\mathbf{y_X} = \\ &= X\Lambda \begin{bmatrix}\mathbf{v_1}^T\mathbf{y}\\ \vdots\\ \mathbf{v_n}^T\mathbf{y}\end{bmatrix} \\ &= \left[\begin{pmatrix}\mathbf{v_1}\end{pmatrix}\lambda_1 \quad \begin{pmatrix}\mathbf{v_2}\end{pmatrix}\lambda_2 \quad \cdots \quad \begin{pmatrix}\mathbf{v_n}\end{pmatrix}\lambda_n\right] \begin{bmatrix}\mathbf{v_1}^T\mathbf{y_E}\\ \vdots\\ \mathbf{v_n}^T\mathbf{y_E}\end{bmatrix} \\ &= \sum_{i=1}^{n} \begin{pmatrix}\mathbf{v_i}\end{pmatrix}\lambda_i(\mathbf{v_i^T})\mathbf{y_E} \end{aligned}$$

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

$$A = \sum_{i=1}^{n} \lambda_i \begin{pmatrix}\mathbf{v_i}\end{pmatrix}(\mathbf{v_i^T})$$

In the end, we wrote the matrix A (that is by hypothesis a symmetric arbitrary matrix) as a linear combination of matrices obtained as $\mathbf{v_i v_i^T}$, 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):

$$A = \lambda_1 \mathbf{v_1 v_1^T} + \sum_{i=2}^{n} \lambda_i \mathbf{v_i v_i^T}$$

and, if we define

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

we have the following:

Claim: If A has

eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$(with $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$)
eigenvectors $v_1, v_2, \ldots, v_n$

then, the matrix A_1 obtained as above has

eigenvalues $0, \lambda_2, \ldots, \lambda_n$(with $|\lambda_2| > \cdots > |\lambda_n|) > 0$
eigenvectors $v_1, v_2, \ldots, v_n$

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 v_1:

$$\begin{aligned} A_1 \mathbf{v_1} &= (A - \lambda_1 \mathbf{v_1 v_1^T})\mathbf{v_1} \\ &= A\mathbf{v_1} - \lambda_1 \mathbf{v_1 v_1^T v_1} \\ &\left[\mathbf{v_1^T v_1} = 1 \text{ by orthonormality of the basis}\right] \\ &= A\mathbf{v_1} - \lambda_1 \mathbf{v_1} \\ &\left[\mathbf{v_1} \text{ is eigenvector of A, with } \lambda_1 \text{ eigenvalue}\right] \\ &= \lambda_1 \mathbf{v_1} - \lambda_1 \mathbf{v_1} = 0 \\ &\Rightarrow A_1 \mathbf{v_1} = 0\mathbf{v_1} \\ &\Rightarrow 0 \text{ is eigenvalue of } A_1, \\ &\text{with corresponding eigenvector } \mathbf{v_1} \end{aligned}$$

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

$$\begin{aligned} & A_1 \mathbf{v_i} = A\mathbf{v_i} - \lambda_1 \mathbf{v_1 v_1^T v_i} = \\ & \left[\mathbf{v_p^T v_q} = 0, \forall p \neq q, \text{because of orthonormality of basis}\right] \\ & \quad = A\mathbf{v_i} \end{aligned}$$

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

Using the just-explained argument, we can define the Deflation Method as follows: it entails subtracting from A a matrix created by multiplying the found eigenvector by its own transposition and scaling it by the found eigenvalue before computing the M largest or smallest eigenvectors of a matrix A one at a time. We just need to calculate the biggest or smallest eigenvalue of the resultant matrix at each stage of the process. Although this approach can soon become unstable because approximation mistakes spread at each step, it can still be used to determine part of a symmetric matrix's greatest or smallest eigenvalues, which is useful for many applications, such as the Spectral Clustering.

2) *Inverse power method*

At this point, we may introduce the Inverse Power Method, which functions as the Deflation Method's "kernel"—that is, the method we employ during each iteration of the Deflation Method in order to identify the matrix's smallest eigenvalue. We shall demonstrate that the Power approach, notably in its Inverse form, offers some great convergence qualities for symmetric matrices, which are the focus of our research. Of course, this is not the only approach one may use as the kernel.

The greatest in module eigenvector (and matching eigenvalue) are calculated using the power approach. Let's examine its operation.
Let A be a real matrix in $\mathbb{R}^{n \times n}$ (not necessarily symmetric), with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$, such that $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n| \geq 0$, and with corresponding eigenvectors $v_1, \ldots, v_n$. Then, any vector $z_0 \in \mathbb{R}^n$ can be written as

$$z_0 = \alpha_1 v_1 + \cdots + \alpha_n v_n$$

Now, define

$$\begin{aligned} z_1 := Az_0 &= \alpha_1 A v_1 + \cdots + \alpha_n A v_n \\ &= \alpha_1 \lambda_1 v_1 + \cdots + \alpha_n \lambda_n v_n \\ &= \lambda_1 \big[\alpha_1 v_1 + \alpha_2 (\frac{\lambda_2}{\lambda_1}) v_2 + \cdots + \alpha_n (\frac{\lambda_n}{\lambda_1}) v_n\big] \\ &\vdots \end{aligned}$$

And so on. So, by induction

$$z_k = \lambda_1^k \big[\alpha_1 v_1 + \alpha_2 (\frac{\lambda_2}{\lambda_1})^k v_n + \cdots + \alpha_n (\frac{\lambda_n}{\lambda_1})^k v_n\big], k \geq 0 \quad (1)$$

Then, we have that

$$\lim_{k \to \infty} z_k = \lambda_1^k \alpha_1 v_1, \quad (2)$$

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

From this we can conclude that, after a certain number of iterations, the vector z_k (that is an arbitrary vector of $\mathbb{R}^n$) has the same direction of v_1, which is the eigenvector of A associated with the eigenvalue λ_1. In other words, we are approximating the eigenvector v_1 (which we don't have) with z_k (that is simply the application of A k times to the random vector z_0).
A problem arises in this procedure: at each iteration, we are finding z_k as

$$z_k = Az_{k-1} = A^k z_0 \approx \lambda_1^k \alpha_1 v_1$$

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 v_1):

$$z_1 \leftarrow \frac{z_1}{||z_1||}$$

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

$$\lim_{k\to\infty} \frac{1}{\lambda_1^k} z_k = \alpha_1 v_1,$$

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

$$\lim_{k\to\infty} \frac{\frac{1}{\lambda_1^{k+1}} z_{k+1}[i]}{\frac{1}{\lambda_1^k} z_k[i]} = \frac{\alpha_1 v_1[i]}{\alpha_1 v_1[i]} = 1.$$

And thus

$$\lim_{k\to\infty} \frac{z_{k+1}[i]}{z_k[i]} = \lambda_1. \tag{4}$$

In other words, we could take the ratio between the component i of the z 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:

$$R_a^k := \frac{z_k^T A z_k}{z_k^T z_k} = \frac{z_k^T z_{k+1}}{z_k^T z_k}$$

We quickly observe that, in contrast to Equation (4), this ratio requires computing a Euclidean norm at each step, making it more computationally expensive than just picking a vector element. Anyhow, we'll examine what benefits this ratio provides for our particular situation.

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

$$z_k = \lambda_1^k \big[\alpha_1 v_1 + \alpha_2 (\frac{\lambda_2}{\lambda_1})^k v_n + \cdots + \alpha_n (\frac{\lambda_n}{\lambda_1})^k v_n\big] = \\ := \lambda_1^k u_k$$

and, equivalently,

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

Then, we can evaluate the following

$$\lim_{k\to\infty} \frac{z_k^T z_{k+1}}{z_k^T z_k} = \frac{\lambda_1^{2k+1}}{\lambda_1^{2k}} \frac{\Big[\alpha_1^2 v_1^T v_1 + O[(\frac{\lambda_2}{\lambda_1})^{k+1}] + O[(\frac{\lambda_2}{\lambda_1})^{2k+1}]\Big]}{\Big[\alpha_1^2 v_1^T v_1 + O[(\frac{\lambda_2}{\lambda_1})^{k}] + O[(\frac{\lambda_2}{\lambda_1})^{2k}]\Big]} = \\ = \lambda_1 \frac{\alpha_1^2 v_1^T v_1}{\alpha_1^2 v_1^T v_1} = \lambda_1$$

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:

$$\text{eigenvalues: } |\lambda_1 - \lambda_1^{(k)}| \le C(\frac{\lambda_2}{\lambda_1})^k, \text{with C} \in \mathbb{R}$$
$$\text{eigenvectors: } ||v_1 - z_k|| \le D(\frac{\lambda_2}{\lambda_1})^k, \text{with D} \in \mathbb{R}$$

As a result, the rate of convergence is determined by a power of the ratio between the greatest and second-largest eigenvalues.
Let's now add the supposition that matrix A is symmetric. Real symmetric matrices have the feature that there is an orthonormal basis of eigenvectors, where the product of two separate eigenvectors is zero, but the product of an eigenvector and itself is one. Therefore, if we suppose that A is a real symmetric matrix, all of the "$v_1^T v_1$" terms used to calculate the Rayleigh quotient's numerator and denominator are 1, whereas all of the "$v_i v_j$ with $i \neq j$" terms 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:

$$\text{eigenvalues: } |\lambda_1 - \lambda_1^{(k)}| \le C(\frac{\lambda_2}{\lambda_1})^{2k}, \text{with C} \in \mathbb{R}$$
$$\text{eigenvectors: } ||v_1 - z_k|| \le D(\frac{\lambda_2}{\lambda_1})^{2k}, \text{with D} \in \mathbb{R}$$

In other words, if A is symmetric, the Power Method's rate of convergence doubles. This is what motivates our decision to use it to complement the Deflation Method in order to identify the fewest eigenvalues of a symmetric matrix.

Before beginning the implementation, there is one more step that must be completed. The Power Method is effective for locating a matrix's biggest eigenvalue, but our difficulty is locating its smallest. We introduce the Inverse Power Method for these reasons.

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

$$Av_i = \lambda_i v_i$$
$$\Big[\text{A symmetric} \implies \exists A^{-1}\Big]$$
$$A^{-1}Av_i = \lambda_i A^{-1} v_i$$
$$\Big[\text{A symmetric} \implies \exists \text{ orthonormal basis}\Big]$$
$$v_i = \lambda_i A^{-1} v_i \iff A^{-1} v_i = \frac{1}{\lambda_i} v_i$$

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 $v^{(k+1)}$ we solve the equivalent equation $Av^{(k+1)} = xv(k)$, instead of having to solve $v^{(k+1)} = A^{-1}v^{(k)}$.

III. Implementation

The inverse_power_method (Algorithm 1) and the deflation_method (Algorithm 2), two of the major functions we implemented, are presented here. We will offer an understanding of the tools we utilised and their structure specifically for the Python implementation.

The starting vector of the **inverse_power_method** function is initialised randomly before a series of iterations begin. Each time, the current vector is normalised before applying the LU factorization to solve the matrix equation $Ax = \lambda x$ (which is determined a priori, outside of the for loop). It should be noted that solving this equation equals determining the eigenvector x linked to the least eigenvalue λ. Once more normalising the outcome, the procedure is carried out repeatedly until convergence is attained or the maximum number of iterations has been used. The closest estimate of the lowest eigenvalue and its related eigenvector is the result of the ”inverse_power_method” function.

As we have shown, the ”inverse_power_method” (which is intended to simply identify the smallest eigenvalue) may iteratively uncover several eigenvalues thanks to the **deflation_method** function. By iteratively deflating the matrix following the computation of each eigenvalue, it is intended to compute many lowest eigenvalues. The symmetric matrix A and the required number of eigenvalues, M, are inputs to the function.The ”inverse power method” is used M times within the function, with each call resulting in the computation of the least eigenvalue. The matrix is deflated by subtracting the outer product of the eigenvector from itself when an eigenvalue and its corresponding eigenvector are produced, thereby eliminating the calculated eigenvalue’s contribution from the matrix. The dimensionality of the matrix is decreased throughout this deflation process, enabling the determination of following eigenvalues. This procedure is continued via the ”deflation method” until M eigenvalues are computed and their associated eigenvectors are acquired.

Algorithm 1 Inverse Power Method for Computing Smallest Eigenvalue and Eigenvector

```
1: function INVERSE_POWER_METHOD($A, max\_iteration = 50, tollerance = 1e-15$)
2:     Input:
3:         $A$: Sparse matrix
4:         $max\_iteration$: Maximum number of iterations (default: 50)
5:         $tollerance$: Tolerance for convergence (default: 1e-15)
6:     Output:
7:         $smallest\_eigenvalue$: Computed smallest eigenvalue
8:         $eigenvector$: Normalized eigenvector corresponding to the smallest eigenvalue
9:
10:    $xn$ ← Create a matrix of zeros with dimensions $(A.shape[0], niterations\_max + 1)$
11:    $xn[:, 0]$ ← Initialize with a random vector of ones with noise
12:    $v\_old \leftarrow xn[:, 0]/\|xn\|$
13:    $lambda\_1\_inv\_new \leftarrow 1$
14:    $LU$ ← Perform LU factorization of $A$
15:    for $k \leftarrow 1$ to $niterations\_max$ do
16:        $lambda\_1\_inv\_old \leftarrow lambda\_1\_inv\_new$
17:        $v\_new$ ← Solve the linear system using LU fact.
18:        $lambda\_1\_inv\_new \leftarrow v\_old^T \cdot v\_new$
19:        $v\_old \leftarrow v\_new/\|v\_new\|$
20:        if $|lambda\_1\_inv\_new - lambda\_1\_inv\_old| < tol$ then
21:            break
22:        end if
23:    end for
24:    return $(1/lambda\_1\_inv\_new, v\_old)$
25: 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* [1] function of the *Scipy* Python library for the symmetric matrix, and using the *scipy.sparse.linalg.eigs* [2] function of the same library for the non-symmetric case. The error is of the order of 10^{-2} for both cases.
2) For the symmetric situation, the convergence is expedited as we can observe that stability is attained fairly instantly. Zooming in on the graph reveals that stability is attained between iterations 5 and 6. On the other hand, we can see graphically how the stability is obtained in

Algorithm 2 Deflation Method for Computing M Smallest Eigenvalues and Eigenvectors

```
1:  function DEFLATION_METHOD($A, max\_eigenvalues, max\_i$
    $50, tollerance = 1e - 15$)
2:      Input:
3:          $A$: Sparse matrix
4:          $max\_eigenvalues$: Number of smallest eigenvalues to compute
5:          $max\_iteration$: Maximum number of iterations for inverse power method (default: 50)
6:          $tollerance$: Tolerance for convergence of inverse power method (default: 1e-15)
7:      Output:
8:          $eigenvalues$: Array of the M smallest eigenvalues
9:          $eigenvectors$: Array of the M corresponding eigenvectors
10:
11:     $eigenvalues$ ← Create an array of zeros with size M
12:     $eigenvectors$ ← Create a matrix of zeros with dimensions $(A.shape[0], M)$
13:
14:     for $i \leftarrow 0$ to $M - 1$ do
15:         $eigenvalue, eigenvector \leftarrow$ INVERSE_POWER_METHOD($A, niterations\_max, tol$)
16:         $eigenvalues[i] \leftarrow eigenvalue$
17:         $eigenvectors[:, i] \leftarrow eigenvector$
18:
19:         Deflation step: Update $A$ by removing the contribution of the computed eigenvector
20:         $A \leftarrow A - eigenvalue \cdot$ outer($eigenvector, eigenvector$)
21:     end for
22:
23:     return $eigenvalues, eigenvectors$
24: end function
```

the non-symmetric scenario at around iteration 15-20. Thus, we can attest to the Power Method's property—the doubling of convergence speed—for symmetric matrices.

We'll evaluate the Deflation Method function next. We explore how the 10 lowest eigenvalues found using the Deflation Method with Inverse Power Method kernel compare to the theoretical eigenvalues using 4 symmetric matrices from the test. The function [2] was also used in this instance to obtain the theoretical eigenvalues.

A strange implementation problem with the deflation method was discovered during the method evaluation. Indeed, it is possible to observe that the 10 smallest eigenvalues of the matrix obtained using our deflation method are all flat near 0, despite the fact that the theoretical values are different.

We looked into this thoroughly and discovered that the instability was a defining feature of the deflation mechanism, not

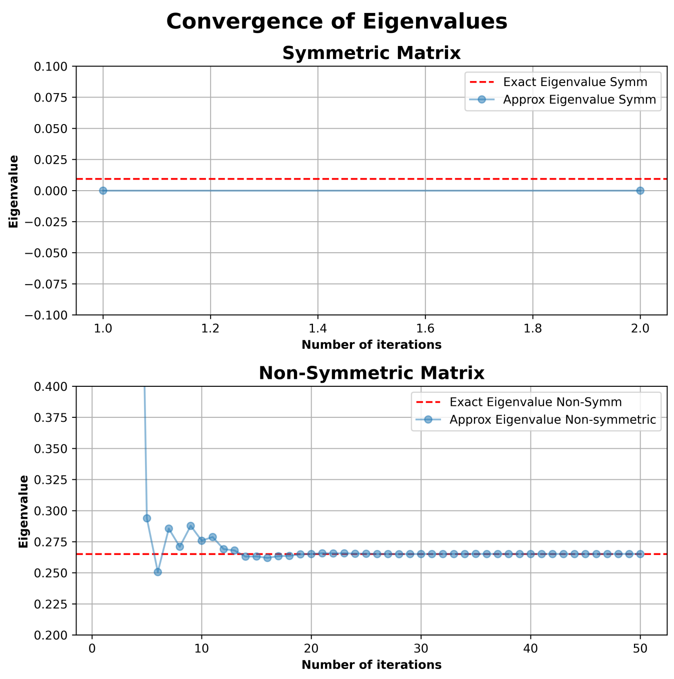

Fig. 1. Inverse Power Convergence A method for calculating the lowest eigenvalue of a matrix with 1000 rows and 1000 columns and 10% sparsity that is both symmetric and nonsymmetric.

a flaw in our implementation.

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

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 this be handled? Instead of using it, we utilise this data to hypothesise that a small amount of "normalisation" could aid in achieving method convergence. We observed that the deflation approach performs significantly better when we scale the amount that we remove from the matrix at each step by a positive random value $\approx 10^3$. Therefore, we choose to employ the quantity as a scaling factor

$$\frac{1}{\hat{\lambda}_1},$$

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 2, we can assess that the deflation method is now converging to the right smallest eigenvalues of A.

V. Conclusion

We summarise our findings to wrap up this little investigation. We began by outlining the Deflation Method and how it may be used to determine a matrix's top M biggest or smallest eigenvalues (and related eigenvectors). Then, as our challenge included calculating the first M eigenvalues of a symmetric matrix, we decided to analyse the Inverse Power Method, which is a solid choice in general. In fact, we have observed that this approach, when used with symmetric matrices, has better convergence features. Then, we put the strategies we had just discussed into practise and evaluated them visually. During this process, we identified a stability problem and suggested a potential solution. Finally, in order to address the first question, we may state that it is possible to quickly compute some of the lowest eigenvalues and eigenvectors of a symmetric real matrix by combining the Deflation Method with the Inverse Power Method. In general, we have observed that the method is sufficiently 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. The number of eigenvalues we can consider reliable varies from application to application.

References

[1] 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.

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

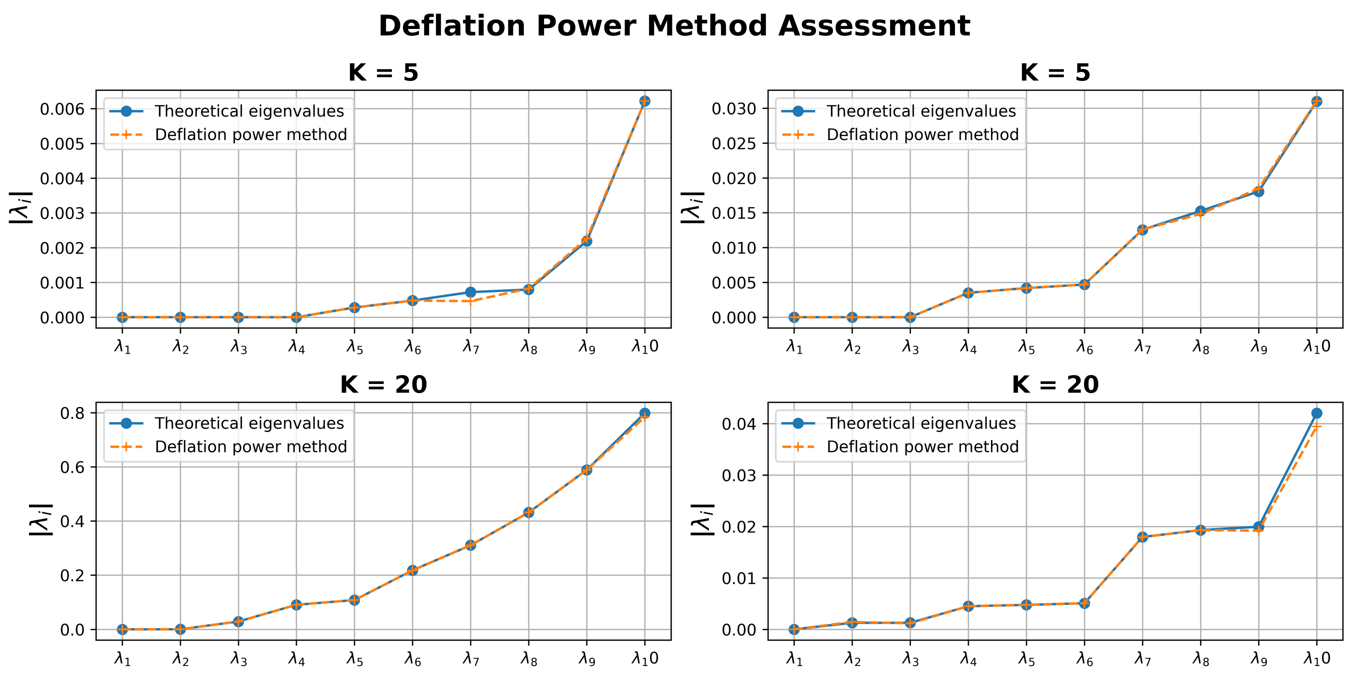

Fig. 2. Theoretical and approximated eigenvalues on four separate big sparse matrices with 1000 rows and 1000 columns are used to evaluate the Deflation Power Method.