Eigenvalue computation for sparse matrices

Massimiliano Ferronato

Dept. Mathematical Methods and Models for Scientific Applications (DMMMSA)

University of Padova

E-mail: ferronat@dmsa.unipd.it

http://www.dmsa.unipd.it/~ferronat

Contents

1 Introduction

The eigenvalue problem for a $(n \times n)$ matrix A relies on the determination of the nontrivial solutions $\boldsymbol{u}$ to:

$$A\boldsymbol{u} = \lambda \boldsymbol{u} \tag{1}$$

with $\lambda \in \mathbb{C}$. The eigenproblem (1) is equivalent to finding the roots of the characteristic equation:

$$\det(A - \lambda I) = 0 \tag{2}$$

Obviously, this is not a viable technique. First, the coefficients of the characteristic equation cannot be computed in a numerically stable way. Second, even if the characteristic equation is accurate enough, the computation of its roots is highly unstable with small perturbations to the coefficients leading to large perturbations on the roots. Therefore, it is necessary to find alternative ways to compute the eigenvalue/eigenvector pairs $(\lambda, \boldsymbol{u})$ of A. Looking at the nature of this problem, however, it is clear that all methods for solving an eigenproblem must be iterative. In fact, the contrary would be in contradiction with the fundamental theorem of Abel-Ruffini that no algorithms with a finite number of operations exist for the computation of the roots of a polynomial with degree larger than 4.

The conditioning of an eigenproblem is related to the way a perturbation on the matrix coefficients impacts on the eigenvalues. The following theorem can be proved:

THEOREM 1 (Bauer-Fike) *Let $A \in \mathbb{R}^{n \times n}$ be a diagonalizable matrix and $A + E$ a perturbation of A, and indicate with P the matrix whose columns are the eigenvectors of A. If ξ is an eigenvalue of $A + E$, then at least one eigenvalue λ of A is such that $|\lambda - \xi| \leq \|P\|\|P^{-1}\|\|E\|$.*

In practice, the theorem of Bauer-Fike says that the conditioning index κ of an eigenproblem is actually the conditioning index of the matrix of its eigenvectors. As a consequence, finding the eigenvalues of a symmetric matrix is generally a well-conditioned problem because P is orthogonal. By contrast, an eigenproblem is ill-conditioned when a few eigenvectors are almost parallel.

There are several different techniques for solving the eigenproblem (1). First of all, it is necessary to answer a few preliminary questions. Is the matrix real symmetric or not? In the former case, only real eigenvalues and eigenvectors are to be computed, not in the latter. Do we need only the extreme eigenvalues, i.e. the largest and the smallest in absolute value? Or do we need some extremal eigenvalues? Or else all the eigenvalues? According to the actual problem, completely different strategies are used. In the following sections first we will address the issue of computing only the largest and the smallest eigenvalue. Then, the problem of computing the full spectrum is considered, in particular for the case of sparse matrices as they typically arise from Finite Element discretized PDEs.

2 Computation of the extreme eigenvalues

In many engineering problems only the largest or the smallest modulus of the eigenvalues of a square matrix is needed. For example, both the largest and the smallest eigenvalues are useful to estimate the convergence properties of an iterative method such as the Preconditioned Conjugate Gradient (PCG). In a structural engineering problem, the smallest eigenvalue of the stiffness matrix represents the first oscillation frequency whose knowledge is needed in several applications.

In the sequel we will assume as usual that the eigenvalues are ordered as follows:

$$|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n| \tag{3}$$

thus indicating with λ_1 and λ_n the largest and the smallest eigenvalue, respectively, and with $\boldsymbol{u}_1$ and $\boldsymbol{u}_n$ the related eigenvectors. We assume also that the n eigenvectors are linearly independent.

2.1 The power method

The oldest and the simplest method for solving eigenvalue problems is the so-called power method. It is also known as single-vector iteration method as it is based on a simple recursion starting from an arbitrary nonzero vector.

Let us call $\boldsymbol{z}_0$ the initial nonzero vector. The power method generates a sequence of vectors $\boldsymbol{z}_k$ using the following recurrence:

$$\boldsymbol{z}_{k+1} = A\boldsymbol{z}_k \tag{4}$$

It can be simply observed that any vector $\boldsymbol{z}_k$ obatined by equation (4) is actually:

$$\boldsymbol{z}_k = A^k \boldsymbol{z}_0 \tag{5}$$

and converges to the eigenvector $\boldsymbol{u}_1$ associated to the dominant eigenvalue. In fact, recalling that under the above hypotheses $\boldsymbol{z}_0$ can be written as a linear combination of the eigenvectors of A, we have:

$$\boldsymbol{z}_k = \sum_{i=1}^{n} c_i A^k \boldsymbol{u}_i \simeq \lambda_1^k c_1 \boldsymbol{u}_1 \tag{6}$$

From equation (6) it follows that $|\lambda_1|$ is readily obtained as:

$$|\lambda_1| = \lim_{k\to\infty} \frac{\|\boldsymbol{z}_{k+1}\|}{\|\boldsymbol{z}_k\|} \tag{7}$$

where $\|\cdot\|$ indicates any vector norm, e.g. the 2-norm or the ∞-norm. The above relationships give rise to the following very simple algorithm for the computation of the dominant eigenvalue of a square matrix A:

ALGORITHM 1: THE POWER METHOD

1. Choose $\boldsymbol{z}_0$
2. DO $k = 0, \ldots$ until convergence
3. $\quad \boldsymbol{y}_{k+1} = A\boldsymbol{z}_k$
4. $\quad \alpha_{k+1} = \|\boldsymbol{y}_{k+1}\|_\infty$
5. $\quad \boldsymbol{z}_{k+1} = \boldsymbol{y}_{k+1}/\alpha_{k+1}$
6. $\quad |\lambda_1|^{(k+1)} = \alpha_{k+1}$
7. END DO

Algorithm 1 is just one of the variants of the power method with the aim at normalizing the norm the newly generated vector $\boldsymbol{z}_{k+1}$. In fact, if no normalization is performed the computation may soon stop because of an overflow error due to the very quick growth of the size of the $\boldsymbol{z}_{k+1}$ components.

The power method has a linear convergence. Looking again at equation (6), the current vector $\boldsymbol{z}_k$ can be also written as:

$$\boldsymbol{z}_k = \lambda_1^k \left[c_1 \boldsymbol{u}_1 + \sum_{i=2}^{n} \left(\frac{\lambda_i}{\lambda_1} \right)^k c_i \boldsymbol{u}_i \right] \tag{8}$$

Equation (8) shows that at step k the error is practically proportional to $(\lambda_2/\lambda_1)^k$, i.e. to the largest term of the neglected sum. In a semi-logarithmic scale, the error decreases proportionally to k with convergence rate equal to $-\log(|\lambda_2|/|\lambda_1|)$. Hence, the larger the separation between the two dominant eigenvalues, the faster the convergence.

In case λ_1 is not simple, the power method still works for determining the dominant eigenvalue but not the associated eigenvectors. Assuming that λ_1 has multeplicity equal to r, at step k the vector $\boldsymbol{z}_k$ will be approximately parallel to a linear combination of $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r$. Equation (6) also shows that theoretically the power method cannot converge to $|\lambda_1|$ if $c_1 = 0$. However, if the iteration proceeds for a sufficiently long time at some step c_1 will not be exactly 0 because of the rounding errors and quickly the parallelism to the eigenvector $\boldsymbol{u}_1$ will resume. Similarly, if A has $s < n$ independent eigenvectors and $\boldsymbol{z}_0$ does not belong to the subspace generated by these eigenvectors, then the power method cannot converge to any dominant eigenvalue. A final remark is needed in case λ_1 is complex. In fact, there are two dominant eigenvalues, λ_1 and its conjugate λ_1^*, and the power method will be characterized by undamped oscillations which do not provide any useful indication on the actual value of $|\lambda_1|$.

The power method can be used for the computation of λ_n, too. A first idea is employing Algorithm 1 with A^{-1} instead of A, exploiting the fact that the largest eigenvalue of A^{-1} is $1/\lambda_n$. The limit of such an approach is that a linear system has to be solved at each step k. Moreover,

a large number of steps is usually required because the convergence rate reads:

$$-\log \frac{|\lambda_{n-1}|}{|\lambda_n|} \simeq 0 \tag{9}$$

An alternative way to exploit the power method for the computation of λ_n relies on properly shifting A. Assuming that λ_1 is known, set $\sigma > \lambda_1$ and build the shifted matrix B:

$$B = \sigma I - A \tag{10}$$

It is easy to observe that the dominant eigenvalue μ_1 of B reads:

$$\mu_1 = \sigma - \lambda_n \tag{11}$$

Using the power method with B readily provides λ_n as the difference between σ and μ_1. Actually, the convergence rate of the shifted power method can be quite slow because:

$$-\log \frac{|\mu_2|}{|\mu_1|} = \log(\sigma - \lambda_n) - \log(\sigma - \lambda_{n-1}) \simeq 0 \tag{12}$$

Moreover, the computation of a small number like λ_n as the difference between much larger numbers can easily lead to inaccurate results.

Finally, the shifted and inverse power methods can be combined together to compute the eigenvalue of A closest to a given scalar σ by applying Algorithm 1 to B^{-1} with B given by equation (10). This strategy, known as *shift-and-invert technique*, is often used in combination with other schemes, such as Krylov subspace methods, to obtain accurate approximations of a few specific eigenvalues.

2.2 Conjugate gradient for the Rayleigh quotient

An efficient technique to compute the smallest eigenvalue of a real symmetric positive definite matrix A relies on using a particular variant of the Conjugate Gradient (CG) method.

Let us define the function $q(\boldsymbol{z}) : \mathbb{R}^n \to \mathbb{R}$ as:

$$q(\boldsymbol{z}) = \frac{\boldsymbol{z}^T A \boldsymbol{z}}{\boldsymbol{z}^T \boldsymbol{z}} \tag{13}$$

The real function $q(\boldsymbol{z})$ defined in (13) is called *Rayleigh quotient*. It can be easily shown that the Rayleigh quotient satisfies the following properties:

1. if $\boldsymbol{u}_i$ is the eigenvector associated to the i-th eigenvalue of A, then $q(\boldsymbol{u}_i) = \lambda_i$;
2. $q(\boldsymbol{z})$ is stationary for any $\boldsymbol{u}_i$;
3. $q(\boldsymbol{z})$ is a limited function.

Property 1 is easily verified using the eigenvalue/eigenvector definition (1):

$$q\left(\boldsymbol{u}_i\right) = \frac{\boldsymbol{u}_i^T A \boldsymbol{u}_i}{\boldsymbol{u}_i^T \boldsymbol{u}_i} = \frac{\lambda_i \boldsymbol{u}_i^T \boldsymbol{u}_i}{\boldsymbol{u}_i^T \boldsymbol{u}_i} = \lambda_i \tag{14}$$

Property 2 means that the gradient of $q(\boldsymbol{z})$ is zero for any eigenvector of A. In fact:

$$q'\left(\boldsymbol{z}\right) = \boldsymbol{g}\left(\boldsymbol{z}\right) = \frac{2}{\boldsymbol{z}^T \boldsymbol{z}} \left[A\boldsymbol{z} - q\left(\boldsymbol{z}\right) \boldsymbol{z}\right] \tag{15}$$

and recalling (14) it follows immediately that $\boldsymbol{g}(\boldsymbol{u}_i) = 0$. To prove property 3 write $\boldsymbol{z}$ as a linear combination of the eigenvectors of A:

$$\boldsymbol{z} = \sum_{i=1}^{n} c_i \boldsymbol{u}_i \tag{16}$$

and use equation (16) in the definition (13) of $q(\boldsymbol{z})$:

$$q\left(\boldsymbol{z}\right) = \frac{\sum_{i=1}^{n} c_i^2 \lambda_i}{\sum_{i=1}^{n} c_i^2} \tag{17}$$

To get (17) the property of the eigenvectors of a real symmetric matrix of being an orthonormal basis of $\mathbb{R}^n$ has been exploited. Now imagine of replacing every λ_i in (17) with λ_n obtaining immediately that $q(\boldsymbol{z}) \geq \lambda_n$. Then replacing every λ_i with λ_1 we finally have:

$$\lambda_n \leq q\left(\boldsymbol{z}\right) \leq \lambda_1 \tag{18}$$

thus proving that the Rayleigh quotient is a limited function. Combining property 3 and equation (18) with property 1 and 2 immediately provides that λ_1 and λ_n are the maximum and the minimum of $q(\boldsymbol{z})$, respectively. Hence the extreme eigenvalues of A can be computed applying to the Rayleigh quotient an optimization method such as the CG.

It is well-known that the CG algorithm for solving a linear system seeks the minimum of a quadratic function Φ equal to:

$$\Phi = \boldsymbol{e}^T A \boldsymbol{e} \tag{19}$$

where $\boldsymbol{e}$ is the error vector. Geometrically, $\Phi = \text{cost}$ represents the equation of an iper-ellipsoid in $\mathbb{R}^n$ with the system solution in its center. The Rayleigh quotient (13) is a quadratic function with a different nature. To have a glimpse of the $q(\boldsymbol{z})$ shape let us consider the cartesian representation for the case $n = 2$ of the equation:

$$q\left(\boldsymbol{z}\right) = c = \text{cost} \tag{20}$$

describing the equipotential curves. Setting:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix} \qquad \boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \tag{21}$$

equation (20) takes on the explicit form:

$$\left(a_{11} - c\right) z_1^2 + 2a_{12} z_1 z_2 + \left(a_{22} - c\right) z_2^2 = 0 \tag{22}$$

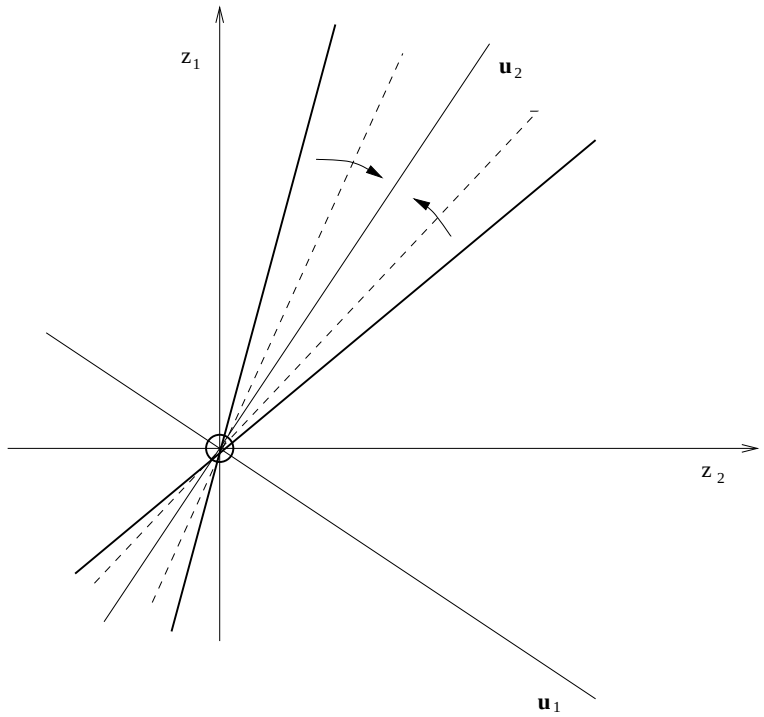

FIGURE 1: *Equipotential curves of* $q(\boldsymbol{z})$ *for* $n = 2$.

Recalling that $\lambda_2 \leq c \leq \lambda_1$, equation (22) can be factorized into the product of two real linear polynomials. Hence, the equipotential curves are nothing but a pair of right lines. As with $c = \lambda_i$ the two lines coincide with the direction of the eigenvector $\boldsymbol{u}_i$, they must be symmetric with respect to the direction of the eigenvectors. In practice, $q(\boldsymbol{z})$ has the shape of a kind of wave symmetric to the origin, with the maximum along $\boldsymbol{u}_1$ and the minimum along $\boldsymbol{u}_2$. Note that the reference origin is a singular point. A sketch of the equipotential curves (22) is shown in Figure 1. For $n > 2$ a similar interpretation is still possible, with the directions $\boldsymbol{u}_i$ of the eigenvectors associated to the intermediate eigenvalues corresponding to local minimum, maximum, or saddle points.

Let us use the CG algorithm to minimize $q(\boldsymbol{z})$. The current eigenvector approximation is given by the recurrence:

$$\boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \alpha_k \boldsymbol{p}_k \tag{23}$$

where the scalar α_k is chosen so as to minimize $q(\boldsymbol{z}_{k+1})$:

$$\frac{\partial}{\partial \alpha_k} \left[\frac{(\boldsymbol{z}_k + \alpha_k \boldsymbol{p}_k)^T A (\boldsymbol{z}_k + \alpha_k \boldsymbol{p}_k)}{(\boldsymbol{z}_k + \alpha_k \boldsymbol{p}_k)^T (\boldsymbol{z}_k + \alpha_k \boldsymbol{p}_k)} \right] = \frac{\partial}{\partial \alpha_k} \left(\frac{\Phi_1}{\Phi_2} \right) = 0 \tag{24}$$

The explicit computation of α_k from equation (24) is a bit heavy. Compute separately the derivatives of Φ_1 and Φ_2 with respect to α_k as:

$$\frac{\partial \Phi_1}{\partial \alpha_k} = 2\boldsymbol{p}_k^T A (\boldsymbol{z}_k + \alpha_k \boldsymbol{p}_k) = 2(a + \alpha_k b) \tag{25}$$

$$\frac{\partial \Phi_2}{\partial \alpha_k} = 2\boldsymbol{p}_k^T (\boldsymbol{z}_k + \alpha_k \boldsymbol{p}_k) = 2(c + \alpha_k d) \tag{26}$$

having set:

$$a = \boldsymbol{p}_k^T A \boldsymbol{z}_k \tag{27}$$

$$b = \boldsymbol{p}_k^T A \boldsymbol{p}_k \tag{28}$$

$$c = \boldsymbol{p}_k^T \boldsymbol{z}_k \tag{29}$$

$$d = \boldsymbol{p}_k^T \boldsymbol{p}_k \tag{30}$$

Now write explicitly Φ_1 and Φ_2:

$$\Phi_1 = \boldsymbol{z}_k^T A \boldsymbol{z}_k + 2\alpha_k \boldsymbol{p}_k^T A \boldsymbol{z}_k + \alpha_k^2 \boldsymbol{p}_k^T A \boldsymbol{p}_k = e + 2\alpha_k a + \alpha_k^2 b \tag{31}$$

$$\Phi_2 = \boldsymbol{z}_k^T \boldsymbol{z}_k + 2\alpha_k \boldsymbol{p}_k^T \boldsymbol{z}_k + \alpha_k^2 \boldsymbol{p}_k^T \boldsymbol{p}_k = m + 2\alpha_k c + \alpha_k^2 d \tag{32}$$

with:

$$e = \boldsymbol{z}_k^T A \boldsymbol{z}_k \tag{33}$$

$$m = \boldsymbol{z}_k^T \boldsymbol{z}_k \tag{34}$$

Developing formally equation (24) yields:

$$\frac{\Phi_2 \dfrac{\partial \Phi_1}{\partial \alpha_k} - \Phi_1 \dfrac{\partial \Phi_2}{\partial \alpha_k}}{\Phi_2^2} = 0 \tag{35}$$

that becomes:

$$\alpha_k^2 (bc - ad) + \alpha_k (bm - de) + (am - ce) = 0 \tag{36}$$

with the aid of equations (25), (26), (31), and (32). The scalar α_k is finally provided by:

$$\alpha_{k1,2} = \frac{(de - bm) \pm \sqrt{\Delta}}{2 (bc - ad)} \tag{37}$$

where:

$$\Delta = (de - bm)^2 - 4 (bc - ad) (am - ce) \tag{38}$$

It can be simply verified that the "+" sign in (37) gives the minimum of $q(\boldsymbol{z}_{k+1})$ and the "-" sign the maximum.

Let us check if the updated vector $\boldsymbol{z}_{k+1}$ is the desired eigenvector $\boldsymbol{u}_n$. The residual vector for problem (1), recalling the property 1 of the Rayleigh quotient, reads:

$$\boldsymbol{r}_{k+1} = A \boldsymbol{z}_{k+1} - q (\boldsymbol{z}_{k+1}) \boldsymbol{z}_{k+1} \tag{39}$$

If the scheme has not converged yet, the search direction $\boldsymbol{p}_k$ is updated using the direction of the gradient of $q(\boldsymbol{z}_{k+1})$:

$$\boldsymbol{p}_{k+1} = \boldsymbol{g}_{k+1} + \beta_k \boldsymbol{p}_k \tag{40}$$

where $\boldsymbol{g}_{k+1} = \boldsymbol{g}(\boldsymbol{z}_{k+1})$. It can be simply observed from equation (15) that $\boldsymbol{g}_{k+1}$ can be updated from $\boldsymbol{r}_{k+1}$ as:

$$\boldsymbol{g}_{k+1} = \frac{2\boldsymbol{r}_{k+1}}{\boldsymbol{z}_{k+1}^T \boldsymbol{z}_{k+1}} \tag{41}$$

The scalar coefficient β_k is computed prescribing the A-orthogonality between the vectors $\boldsymbol{p}_k$ and $\boldsymbol{p}_{k+1}$. Similarly to the CG method for linear systems we have:

$$\beta_k = -\frac{\boldsymbol{g}_{k+1}^T A \boldsymbol{p}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k} \tag{42}$$

When the residual norm falls below a user-specified tolerance $\boldsymbol{z}_{k+1} \simeq \boldsymbol{u}_n$ and $\lambda_n \simeq q(\boldsymbol{z}_{k+1})$. The CG algorithm for the Rayleigh quotient is therefore the following:

ALGORITHM 2: CONJUGATE GRADIENT FOR THE RAYLEIGH QUOTIENT

1. Choose $\boldsymbol{z}_0$ and compute $\boldsymbol{r}_0 = A\boldsymbol{z}_0 - q(\boldsymbol{z}_0)\boldsymbol{z}_0$
2. Set $\boldsymbol{p}_0 = \boldsymbol{r}_0$
3. DO $k = 0, \ldots$ until convergence
4. $\quad a = \boldsymbol{p}_k^T A \boldsymbol{z}_k$
5. $\quad b = \boldsymbol{p}_k^T A \boldsymbol{p}_k$
6. $\quad c = \boldsymbol{p}_k^T \boldsymbol{z}_k$
7. $\quad d = \boldsymbol{p}_k^T \boldsymbol{p}_k$
8. $\quad e = \boldsymbol{z}_k^T A \boldsymbol{z}_k$
9. $\quad m = \boldsymbol{z}_k^T \boldsymbol{z}_k$
10. $\quad \Delta = (de - bm)^2 - 4(bc - ad)(am - ce)$
11. $\quad \alpha_k = [(de - bm) + \sqrt{\Delta}]/2(bc - ad)$
12. $\quad \boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \alpha_k \boldsymbol{p}_k$
13. $\quad \boldsymbol{r}_{k+1} = A\boldsymbol{z}_{k+1} - q(\boldsymbol{z}_{k+1})\boldsymbol{z}_{k+1}$
14. $\quad \boldsymbol{g}_{k+1} = 2\boldsymbol{r}_{k+1}/\boldsymbol{z}_{k+1}^T \boldsymbol{z}_{k+1}$
15. $\quad \beta_k = -\boldsymbol{g}_{k+1}^T A \boldsymbol{p}_k / b$
16. $\quad \boldsymbol{p}_{k+1} = \boldsymbol{g}_{k+1} + \beta_k \boldsymbol{p}_k$
17. END DO

Theoretically, if $\boldsymbol{z}_0$ is orthogonal to $\boldsymbol{u}_n$ all vectors $\boldsymbol{z}_k$ are too, and no convergence is possible to λ_n. In practice, the rounding errors soon generate a non zero component of $\boldsymbol{z}_k$ along $\boldsymbol{u}_n$ allowing for convergence anyway. In case of difficulties in the recognition of the desired eigenvector a different initial guess $\boldsymbol{z}_0$ can be simply selected. The probability that even a second arbitrary vector is orthogonal to $\boldsymbol{u}_n$ is practically zero. Note also that using the "-" sign in step 11 for the computation of α_k would lead to the maximum of $q(\boldsymbol{z})$, i.e. λ_1. However, Algorithm 2 to compute the dominant eigenvalue is much more expensive than Algorithm 1 and so it is not

usually employed for such an aim.

Let us analyze the convergence of Algorithm 2. At step k the current vector $\boldsymbol{z}_k$ can be interpreted as the sum of $\boldsymbol{u}_n$ and some error vector $\boldsymbol{e}$:

$$\boldsymbol{z}_k = \boldsymbol{u}_n + \boldsymbol{e} \tag{43}$$

Writing the Taylor expansion of $q(\boldsymbol{z}_k)$ around $\boldsymbol{u}_n$ and truncating it at the second-order terms yields:

$$q\left(\boldsymbol{z}_k\right) \simeq q\left(\boldsymbol{u}_n\right) + \boldsymbol{e}^T q'\left(\boldsymbol{u}_n\right) + \frac{1}{2}\boldsymbol{e}^T q''\left(\boldsymbol{u}_n\right)\boldsymbol{e} = \lambda_n + \frac{1}{2}\boldsymbol{e}^T H\left(\boldsymbol{u}_n\right)\boldsymbol{e} \tag{44}$$

where the second-order derivative of $q(\boldsymbol{z})$ is actually the Hessian matrix $H(\boldsymbol{z})$. Equation (44) shows that locally the minimization of the Rayleigh quotient corresponds to minimizing the quadratic form of the Hessian of $q(\boldsymbol{z})$. Recalling equation (15), matrix $H(\boldsymbol{z})$ reads:

$$H\left(\boldsymbol{z}\right) = \frac{2}{\boldsymbol{z}^T\boldsymbol{z}}\left[A - q\left(\boldsymbol{z}\right)I - \boldsymbol{g}\left(\boldsymbol{z}\right)\boldsymbol{z}^T - \boldsymbol{z}\boldsymbol{g}^T\left(\boldsymbol{z}\right)\right] \tag{45}$$

Computed for $\boldsymbol{z} = \boldsymbol{u}_n$, equation (45) provides:

$$H\left(\boldsymbol{u}_n\right) = 2\left(A - \lambda_n I\right) \tag{46}$$

with the eigenvector $\boldsymbol{u}_n$ normalized in such a way that $\boldsymbol{u}_n^T\boldsymbol{u}_n = 1$. It is well-known that the convergence of CG for minimizing a quadratic form depends on the spectral conditioning index of the matrix that generates the quadratic function. Setting μ_1 and μ_n the largest and the smallest eigenvalue of $H(\boldsymbol{u}_n)$, respectively, it is readily verified that:

$$\mu_1 = 2\left(\lambda_1 - \lambda_n\right) \tag{47}$$

$$\mu_n = 2\left(\lambda_{n-1} - \lambda_n\right) \tag{48}$$

Hence the spectral conditioning number of $H(\boldsymbol{u}_n)$ reads:

$$\kappa\left[H\left(\boldsymbol{u}_n\right)\right] = \frac{\lambda_1 - \lambda_n}{\lambda_{n-1} - \lambda_n} \tag{49}$$

As typically $\lambda_{n-1} \simeq \lambda_n$, the convergence of Algorithm 2 can be very slow. Therefore, the CG for the Rayleigh quotient must be preconditioned in some sense in order to accelerate its convergence rate. We use the following variable change:

$$\boldsymbol{y} = X\boldsymbol{z} \tag{50}$$

where X is a symmetric non singular matrix. With equation (50) the Rayleigh quotient becomes:

$$q_p\left(\boldsymbol{y}\right) = \frac{\boldsymbol{y}^T X^{-1} A X^{-1}\boldsymbol{y}}{\boldsymbol{y}^T X^{-1} X^{-1}\boldsymbol{y}} = \frac{\boldsymbol{y}^T G\boldsymbol{y}}{\boldsymbol{y}^T K^{-1}\boldsymbol{y}} \tag{51}$$

where $G = X^{-1}AX^{-1}$ and $K^{-1} = X^{-1}X^{-1}$. Note that the new Rayleigh quotient in equation (51) satisfies again the properties 1, 2, and 3 with λ_i an eigenvalue of the generalized eigenproblem:

$$G\boldsymbol{w} = \lambda K^{-1}\boldsymbol{w} \tag{52}$$

It is readily verified that:

$$KG = XXX^{-1}AX^{-1} = XAX^{-1} \tag{53}$$

Being the matrix KG obtained by a similitude transformation of A, the eigenvalues of (52) are the same as A and minimizing $q(\boldsymbol{z})$ in (13) provides the same result as minimizing $q_p(\boldsymbol{y})$ in (51). Writing Algorithm 2 for equation (51) and using the variable change (50) provides the Preconditioned CG for the Rayleigh quotient:

ALGORITHM 3: PRECONDITIONED CONJUGATE GRADIENT FOR THE RAYLEIGH QUOTIENT

1. Choose $\boldsymbol{z}_0$ and compute $\boldsymbol{r}_0 = A\boldsymbol{z}_0 - q(\boldsymbol{z}_0)\boldsymbol{z}_0$
2. Set $\boldsymbol{p}_0 = K^{-1}\boldsymbol{r}_0$
3. DO $k = 0, \ldots$ until convergence
4. $\quad a = \boldsymbol{p}_k^T A \boldsymbol{z}_k$
5. $\quad b = \boldsymbol{p}_k^T A \boldsymbol{p}_k$
6. $\quad c = \boldsymbol{p}_k^T \boldsymbol{z}_k$
7. $\quad d = \boldsymbol{p}_k^T \boldsymbol{p}_k$
8. $\quad e = \boldsymbol{z}_k^T A \boldsymbol{z}_k$
9. $\quad m = \boldsymbol{z}_k^T \boldsymbol{z}_k$
10. $\quad \Delta = (de - bm)^2 - 4(bc - ad)(am - ce)$
11. $\quad \alpha_k = [(de - bm) + \sqrt{\Delta}]/2(bc - ad)$
12. $\quad \boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \alpha_k \boldsymbol{p}_k$
13. $\quad \boldsymbol{r}_{k+1} = A\boldsymbol{z}_{k+1} - q(\boldsymbol{z}_{k+1})\boldsymbol{z}_{k+1}$
14. $\quad \boldsymbol{g}_{k+1} = 2\boldsymbol{r}_{k+1}/\boldsymbol{z}_{k+1}^T \boldsymbol{z}_{k+1}$
15. $\quad \beta_k = -\boldsymbol{g}_{k+1}^T K^{-1} A \boldsymbol{p}_k / b$
16. $\quad \boldsymbol{p}_{k+1} = K^{-1}\boldsymbol{g}_{k+1} + \beta_k \boldsymbol{p}_k$
17. END DO

Note the exact parallelism with the PCG algorithm for linear systems. The matrix K^{-1} is called preconditioner.

To have some indications on how choosing K^{-1}, let us compute the Hessian matrix for $q_p(\boldsymbol{y})$. Following equation (45) we get:

$$H_p(\boldsymbol{y}) = \frac{2}{\boldsymbol{y}^T K^{-1} \boldsymbol{y}} \left[G - q_p(\boldsymbol{y}) K^{-1} - K^{-1} \boldsymbol{y} \boldsymbol{g}_p^T(\boldsymbol{y}) - \boldsymbol{g}_p(\boldsymbol{y}) \boldsymbol{y}^T K^{-1} \right] \tag{54}$$

which calculated for the eigenvector $\boldsymbol{w}_n$ associated to λ_n provides:

$$H_p(\boldsymbol{w}_n) = \frac{2}{\boldsymbol{w}_n^T K^{-1} \boldsymbol{w}_n} \left(G - \lambda_n K^{-1} \right) \tag{55}$$

Notice that the matrix $(G - \lambda_n K^{-1})$ is similar to $(A - \lambda_n I)K^{-1}$. In fact:

$$X\left(G - \lambda_n K^{-1}\right) X^{-1} = XX^{-1}AX^{-1}X^{-1} - \lambda_n XX^{-1}X^{-1}X^{-1} = \left(A - \lambda_n I\right) K^{-1} \tag{56}$$

Therefore we have to choose K^{-1} such that the spectral conditioning number of $H_p(\boldsymbol{w}_n)$ is much less than that in equation (49). Assume for instance that $K^{-1} = A^{-1}$ and that $\boldsymbol{w}_n$ is normalized in such a way that $\boldsymbol{w}_n^T K^{-1} \boldsymbol{w}_n = 1$. The extreme eigenvalues $\mu_{p,1}$ and $\mu_{p,n}$ of $H_p(\boldsymbol{w}_n)$ now read:

$$\mu_{p,1} = 2\left(1 - \lambda_n/\lambda_1\right) \tag{57}$$

$$\mu_{p,n} = 2\left(1 - \lambda_n/\lambda_{n-1}\right) \tag{58}$$

hence the new spectral conditioning number becomes:

$$\kappa\left[H_p\left(\boldsymbol{w}_n\right)\right] = \frac{\lambda_{n-1}}{\lambda_1}\frac{\lambda_1 - \lambda_n}{\lambda_{n-1} - \lambda_n} = \frac{\lambda_{n-1}}{\lambda_1}\kappa\left[H\left(\boldsymbol{u}_n\right)\right] \tag{59}$$

In other words, with $K^{-1} = A^{-1}$ the convergence of Algorithm 3 is much faster than that of Algorithm 2 because of equation (59). As choosing exactly $K^{-1} = A^{-1}$ is not feasible, K^{-1} should resemble in some way A^{-1} with the computation of its product by a vector as cheap as possible. A usual strategy is based on using the same preconditioners as with PCG for linear systems. Popular choices are a diagonal approximation of A or an incomplete Choleski factorization.

3 Computation of all eigenvalues

Computing the full spectrum of large sparse matrices can be a very difficult task. Basically, the main idea of any algorithm is to transform the native matrix A into a similar diagonal or triangular matrix whose eigenvalues can be trivially computed. This operation turns out to be particularly efficient when the transformation matrices are orthogonal. In fact, denoting by Q_k an orthogonal square matrix the transformation:

$$A_{k+1} = Q_k^T A_k Q_k \tag{60}$$

generates a new matrix A_{k+1} with the same eigenvalues as A_k. Starting from $A_0 = A$, the objective is choosing Q_k in such a way that A_{k+1} is a matrix whose spectrum computation is easier than A_k.

Several different algorithms exist depending on the choice of Q_k, which on its turn is related to the possible symmetry of A and its sparsity degree. In the following sections we will consider only the QR method, which is probably the most widespread technique for computing the full spectrum of a medium-size matrix, and the Lanczos algorithm for symmetric sparse matrices. Finally a few hints are given on the Krylov subspace methods which belong to one of the most important classes of algorithms for the computation of the eigenvalues and eigenvectors of large sparse matrices.

3.1 The QR method

The QR method is based on the recurrent similitude transformation (60) where Q_k is obtained by the QR decomposition of A_k. Let us prove that any non singular square matrix A can be decomposed as the product of an orthogonal matrix Q and an upper triangular matrix R. Denoting by $\boldsymbol{q}_i$ and $\boldsymbol{a}_i$ the i-th column of Q and A, respectively, the proposition above is proved if we can found an algorithm allowing for the univocal determination of Q and R such that:

$$QR = \begin{bmatrix} \boldsymbol{q}_1 & \boldsymbol{q}_2 & \cdots & \boldsymbol{q}_n \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_n \end{bmatrix} = A \tag{61}$$

Multiplying Q by the first column of R must provide the first column of A. From equation (61) this is readily obtained as:

$$r_{11}\,\boldsymbol{q}_1 = \boldsymbol{a}_1 \tag{62}$$

As the vectors $\boldsymbol{q}_i$ must be an orthonormal basis such that:

$$\boldsymbol{q}_i^T\,\boldsymbol{q}_j = \delta_{ij} \tag{63}$$

with δ_{ij} the Kronecker delta, the 2-norm of $\boldsymbol{q}_1$ is 1, hence $r_{11} = \|\boldsymbol{a}_1\|_2$. It follows immediately that $\boldsymbol{q}_1 = \boldsymbol{a}_1/\|\boldsymbol{a}_1\|_2$. The second column of A is obtained multiplying Q by the second column of R, i.e.:

$$r_{12}\,\boldsymbol{q}_1 + r_{22}\,\boldsymbol{q}_2 = \boldsymbol{a}_2 \tag{64}$$

Premultiplying both sides of equation (64) by $\boldsymbol{q}_1^T$ and using the orthonormality condition (63) yields:

$$r_{12} = \boldsymbol{q}_1^T\,\boldsymbol{a}_2 \tag{65}$$

The 2-norm of $\boldsymbol{q}_2$ must be 1, hence r_{22} is nothing but:

$$r_{22} = \|\boldsymbol{a}_2 - r_{12}\,\boldsymbol{q}_1\|_2 \tag{66}$$

Finally $\boldsymbol{q}_2$ can be computed explicitly from equation (64) because all other terms are known:

$$\boldsymbol{q}_2 = \frac{1}{r_{22}}\,(\boldsymbol{a}_2 - r_{12}\,\boldsymbol{q}_1) \tag{67}$$

The above reasoning can be easily extended to the third column of A. Premultiplying the equation:

$$r_{13}\,\boldsymbol{q}_1 + r_{23}\,\boldsymbol{q}_2 + r_{33}\,\boldsymbol{q}_3 = \boldsymbol{a}_3 \tag{68}$$

by $\boldsymbol{q}_1^T$ and $\boldsymbol{q}_2^T$ the coefficients r_{13} and r_{23} are easily obtained. Then, r_{33} is the norm of the right-hand side vector and $\boldsymbol{q}_3$ can be computed. Thus the following algorithm for the QR decomposition of A can be inferred:

ALGORITHM 4: QR DECOMPOSITION

1. $r_{11} = \|\boldsymbol{a}_1\|_2$
2. $\boldsymbol{q}_1 = \boldsymbol{a}_1 / r_{11}$
3. DO $j = 2, \ldots, n$
4. DO $i = 1, \ldots, j-1$
5. $r_{ij} = \boldsymbol{q}_i^T \boldsymbol{a}_j$
6. END DO
7. $\boldsymbol{t}_j = \boldsymbol{a}_j - \sum_{k=1}^{j-1} r_{kj} \boldsymbol{q}_k$
8. $r_{jj} = \|\boldsymbol{t}_j\|_2$
9. $\boldsymbol{q}_j = \boldsymbol{t}_j / r_{jj}$
10. END DO

Algorithm 4 can exhibit a breakdown whenever $r_{jj} = 0$, i.e. $\boldsymbol{t}_j$ is the zero vector. This would imply that:

$$\boldsymbol{a}_j = \sum_{k=1}^{j-1} r_{kj} \boldsymbol{q}_k \tag{69}$$

Recall that $\boldsymbol{q}_k$ is a linear combination of the first k columns of A. Therefore equation (69) means that the first j columns of A are linearly dependent. However, this contradicts the hypothesis that A is nonsingular and Algorithm 4 can never abort in exact arithmetic.

The Algorithm 4 actually implements the Gram-Schmidt orthogonalization process. However, the rounding errors arising in the computer floating-point arithmetic soon prevents the generated $\boldsymbol{q}_j$ vectors from being exactly orthonormal, with very small r_{jj} values yielding an unstable numerical behavior. To cope with such difficulty the more stable Modified Gram-Schmidt (MGS) procedure is usually preferred. Once the first $r_{1,j}$ value is computed at step j, it is immediately used to calculate the vector $\boldsymbol{t}_j^{(1)}$:

$$\boldsymbol{t}_j^{(1)} = \boldsymbol{a}_j - \boldsymbol{q}_1^T \boldsymbol{a}_j \boldsymbol{q}_1 \tag{70}$$

Then $\boldsymbol{t}_j^{(1)}$ is projected onto $\boldsymbol{q}_2$ and used to compute $\boldsymbol{t}_j^{(2)}$:

$$\boldsymbol{t}_j^{(2)} = \boldsymbol{t}_j^{(1)} - \boldsymbol{q}_2^T \boldsymbol{t}_j^{(1)} \boldsymbol{q}_2 \tag{71}$$

The procedure continues up to the last available vector $\boldsymbol{t}_j^{(j-1)}$:

$$\boldsymbol{t}_j^{(j-1)} = \boldsymbol{t}_j^{(j-2)} - \boldsymbol{q}_{j-1}^T \boldsymbol{t}_j^{(j-2)} \boldsymbol{q}_{j-1} \tag{72}$$

which coincides with $\boldsymbol{t}_j$ obtained in Algorithm 4. In fact, recalling equation (70) $\boldsymbol{t}_j^{(2)}$ reads:

$$\begin{aligned} \boldsymbol{t}_j^{(2)} &= \boldsymbol{a}_j - \boldsymbol{q}_1^T \boldsymbol{a}_j \boldsymbol{q}_1 - \boldsymbol{q}_2^T \boldsymbol{a}_j \boldsymbol{q}_2 + \boldsymbol{q}_1^T \boldsymbol{a}_j \boldsymbol{q}_2^T \boldsymbol{q}_1 \boldsymbol{q}_1 \\ &= \boldsymbol{a}_j - \sum_{k=1}^{2} \boldsymbol{q}_k^T \boldsymbol{a}_j \boldsymbol{q}_k = \boldsymbol{a}_j - \sum_{k=1}^{2} r_{kj} \boldsymbol{q}_k \end{aligned} \tag{73}$$

where the final right-hand side follows from the orthogonality between $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$. Equation (73) can be easily generalized to $\boldsymbol{t}_j^{(j-1)}$ providing the wanted result.

The MGS procedure is mathematically equivalent to the standard Gram-Schmidt algorithm, but is numerically much more stable. The Modified QR decomposition algorithm reads:

ALGORITHM 5: MODIFIED QR DECOMPOSITION

1. $r_{11} = \|\boldsymbol{a}_1\|_2$
2. $\boldsymbol{q}_1 = \boldsymbol{a}_1 / r_{11}$
3. DO $j = 2, \ldots, n$
4. $\quad \boldsymbol{t}_j^{(0)} = \boldsymbol{a}_j$
5. $\quad$ DO $i = 1, \ldots, j-1$
6. $\quad\quad r_{ij} = \boldsymbol{q}_i^T \boldsymbol{t}_j^{(i-1)}$
7. $\quad\quad \boldsymbol{t}_j^{(i)} = \boldsymbol{t}_j^{(i-1)} - r_{ij}\,\boldsymbol{q}_i$
8. $\quad$ END DO
9. $\quad r_{jj} = \|\boldsymbol{t}_j^{(j-1)}\|_2$
10. $\quad \boldsymbol{q}_j = \boldsymbol{t}_j^{(j-1)} / r_{jj}$
11. END DO

Now we can use the current orthogonal matrix Q_k derived from the QR decomposition of A_k to compute the similar matrix A_{k+1} with equation (60). As $A_k = Q_k R_k$ we have immediately:

$$A_{k+1} = Q_k^T Q_k R_k Q_k = R_k Q_k \tag{74}$$

All matrices A_{k+1} obtained by equation (74) have the same eigenvalues as A. It can be proved that:

$$\lim_{k\to\infty} A_{k+1} = U \tag{75}$$

where U is an upper tridiagonal matrix whose eigenvalues are trivially the diagonal terms. In particular, the following relationships hold true:

$$\lim_{k\to\infty} Q_k = I \tag{76}$$

$$\lim_{k\to\infty} R_k = U \tag{77}$$

The final algorithm for the QR method is therefore the following:

ALGORITHM 6: THE QR METHOD

1. Set $A_0 = A$
2. DO $k = 0, \ldots$ until convergence
3. Compute Q_k and R_k such that $A_k = Q_k R_k$
4. $A_{k+1} = R_k Q_k$
5. END DO

The convergence of the QR method is generally quite fast and this is why the vast majority of the commercial routines for the computation of all the eigenvalues of a square matrix is based on it. For a general matrix, however, Algorithm 4 for the QR decomposition of A_k can be quite heavy. It may become very efficient if the native matrix is tridiagonal, with step 7 reducing to a simple three-term recurrence and matrix Q_k having the Hessemberg form. Therefore, the QR method is often employed after the native matrix A has been reduced to a tridiagonal form by appropriate similar transformations.

3.2 The Lanczos method for symmetric matrices

The basic idea of the Lanczos method relies on the use of an orthogonal similitude relationship transforming A into a tridiagonal matrix. It has been developed for symmetric matrices and is particularly efficient for sparse patterns.

Let us generate a sequence of vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ forming an orthonormal basis of $\mathbb{R}^n$, i.e. satisfying condition (63). Assuming that the first k vectors are already available, the k+1-th vector can be generated using the following recurrence:

$$\hat{\boldsymbol{v}}_{k+1} = A\boldsymbol{v}_k - \sum_{j=1}^{k} h_{jk}\boldsymbol{v}_j \tag{78}$$

The new vector $\hat{\boldsymbol{v}}_{k+1}$ must be orthogonal to any $\boldsymbol{v}_j$, with $j \leq k$:

$$\boldsymbol{v}_j^T \hat{\boldsymbol{v}}_{k+1} = \boldsymbol{v}_j^T A\boldsymbol{v}_k - h_{jk}\boldsymbol{v}_j^T \boldsymbol{v}_j = 0 \qquad j = 1, \ldots, k \tag{79}$$

Recalling that $\boldsymbol{v}_j^T \boldsymbol{v}_j = 1$, equation (79) leads to the definition of h_{jk} as:

$$h_{jk} = \boldsymbol{v}_j^T A\boldsymbol{v}_k \qquad j = 1, \ldots, k \tag{80}$$

Generally, $\hat{\boldsymbol{v}}_{k+1}$ computed by equation (78) has not a unitary 2-norm. Therefore, the new vector $\boldsymbol{v}_{k+1}$ of the orthonormal sequence $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ is nothing but:

$$\boldsymbol{v}_{k+1} = \frac{\hat{\boldsymbol{v}}_{k+1}}{\|\hat{\boldsymbol{v}}_{k+1}\|_2} \tag{81}$$

Starting from an initial arbitrary vector $\boldsymbol{v}_1$ with unitary 2-norm, equations (78), (80) and (81) iterated up to the n-th vector provide the desired orthonormal basis of $\mathbb{R}^n$. The n+1-th vector

will be obviously zero as it turns out to be simultaneously orthogonal to n indepedent vectors of $\mathbb{R}^n$.

Actually, the above relationships can be further simplified exploiting the symmetry of A. First, it can be easily shown that $h_{jk} = 0$ for any $j < k - 1$. In fact, h_{jk} is also equal to:

$$h_{jk} = \boldsymbol{v}_j^T A \boldsymbol{v}_k = \boldsymbol{v}_k^T A \boldsymbol{v}_j = \boldsymbol{v}_k^T \left(\|\hat{\boldsymbol{v}}_{j+1}\|_2 \boldsymbol{v}_{j+1} + \sum_{i=1}^{j} h_{ij} \boldsymbol{v}_i \right) \tag{82}$$

It is readily verified that the right-hand side of equation (82) is not zero only for $j = k - 1$ and $j = k$ because of the orthogonality condition characterizing the sequence of vectors $\boldsymbol{v}_i$. Second, the norm of the newly generated vector $\hat{\boldsymbol{v}}_{k+1}$ can be computed as a new coefficient $h_{k,k+1}$. By its definition, we have:

$$h_{k,k+1} = \boldsymbol{v}_k^T A \boldsymbol{v}_{k+1} = \boldsymbol{v}_{k+1}^T A \boldsymbol{v}_k = \boldsymbol{v}_{k+1}^T \left(\|\hat{\boldsymbol{v}}_{k+1}\|_2 \boldsymbol{v}_{k+1} + h_{k,k} \boldsymbol{v}_k + h_{k-1,k} \boldsymbol{v}_{k-1} \right) = \|\hat{\boldsymbol{v}}_{k+1}\|_2 \tag{83}$$

The coefficients $h_{k,k}$, $h_{k-1,k}$, and $h_{k,k+1}$ are classically denoted as α_k, β_k, and β_{k+1}. Thus, the relation (78) reduces to a simple and efficient three-term recurrence:

$$\beta_{k+1} \boldsymbol{v}_{k+1} = A \boldsymbol{v}_k - \alpha_k \boldsymbol{v}_k - \beta_k \boldsymbol{v}_{k-1} \tag{84}$$

The recurrence (84) can be iterated up to $k = n$. As previously observed, $\boldsymbol{v}_{n+1} = 0$ and the above algorithm aborts. Collecting all vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ and setting $\beta_1 = 0$, from (84) we have:

$$\begin{aligned}
A\boldsymbol{v}_1 &= \alpha_1 \boldsymbol{v}_1 + \beta_2 \boldsymbol{v}_2 \\
A\boldsymbol{v}_2 &= \beta_2 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \beta_3 \boldsymbol{v}_3 \\
&\vdots \\
A\boldsymbol{v}_k &= \beta_k \boldsymbol{v}_{k-1} + \alpha_k \boldsymbol{v}_k + \beta_{k+1} \boldsymbol{v}_{k+1} \\
&\vdots \\
A\boldsymbol{v}_{n-1} &= \beta_{n-1} \boldsymbol{v}_{n-2} + \alpha_{n-1} \boldsymbol{v}_{n-1} + \beta_n \boldsymbol{v}_n \\
A\boldsymbol{v}_n &= \beta_n \boldsymbol{v}_{n-1} + \alpha_n \boldsymbol{v}_n
\end{aligned} \tag{85}$$

Writing equations (85) in matrix form yields:

$$AV_n = V_n T_n \tag{86}$$

where V_n is the matrix whose columns are the vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ and:

$$T_n = \begin{bmatrix}
\alpha_1 & \beta_2 & & & \\
\beta_2 & \alpha_2 & \beta_3 & & \\
& & \cdots & & \\
& & \beta_{n-1} & \alpha_{n-1} & \beta_n \\
& & & \beta_n & \alpha_n
\end{bmatrix} \tag{87}$$

As the vectors $\boldsymbol{v}_i$ are an orthonormal basis of $\mathbb{R}^n$, the matrix V_n is orthogonal. Hence premultiplying equation (86) by V_n^T we finally get:

$$T_n = V_n^T A V_n \tag{88}$$

which implies that T_n is a tridiagonal matrix with the same spectrum as A. Using the QR method with T_n allows for the computation of all the eigenvalues of A. On summary, the Lanczos algorithm for symmetric matrices is the following:

ALGORITHM 7: THE LANCZOS METHOD

1. Choose $\boldsymbol{v}_1$ such that $\|\boldsymbol{v}_1\|_2 = 1$
2. Set $\beta_1 = 1$ and $\boldsymbol{v}_0 = 0$
3. $\alpha_1 = \boldsymbol{v}_1^T A \boldsymbol{v}_1$
4. DO $k = 1, \ldots, n-1$
5. $\quad \hat{\boldsymbol{v}}_{k+1} = A\boldsymbol{v}_k - \alpha_k \boldsymbol{v}_k - \beta_k \boldsymbol{v}_{k-1}$
6. $\quad \beta_{k+1} = \|\hat{\boldsymbol{v}}_{k+1}\|_2$
7. $\quad \boldsymbol{v}_{k+1} = \hat{\boldsymbol{v}}_{k+1}/\beta_{k+1}$
8. $\quad \alpha_{k+1} = \boldsymbol{v}_{k+1}^T A \boldsymbol{v}_{k+1}$
9. END DO
10. Compute T_n
11. Use the QR method with T_n

In case the initial vector $\boldsymbol{v}_1$ has s zero components along s eigenvectors of A, theoretically $\boldsymbol{v}_{m+1} = 0$ with $m = n - s$. The matrix T_m, therefore, has order m and does not have all the eigenvalues of A. Such an inconvenience can be avoided simply changing the initial vector. The same difficulty can arise if A has eigenvalues with a multiplicity larger than 1.

The Lanczos algorithm is very efficient since we need to save only three vectors and matrix T_n is readily computed just collecting all the coefficients α_k and β_k. Its main limit, however, is that the orthogonality of vectors $\boldsymbol{v}_i$ is actually guaranteed only in exact arithmetic. With computer arithmetic the exact orthogonality of these vectors is generally observed only at the beginning of the process and the loss of orthogonality can occur quite rapidly. As a consequence, the matrix T_n is not exactly similar to A with some of its eigenvalues not belonging to the native matrix. Moreover, $\boldsymbol{v}_{n+1}$ will not be zero and the DO instruction in Algorithm 7 can be extended to k values larger than $n-1$. The resulting matrix T_m, with $m > n$, has more eigenvalues than A and at least $m - n$ will be *spurious* eigenvalues. A way for addressing the difficulties caused by the rounding errors is therefore to use Algorithm 7 as an iterative method where m progressively increases. The iterations will be stopped when n eigenvalues of T_m stagnate, i.e. they do not change comparing

the spectrum of T_{m-1} with that of T_m. This is due to the fact that the spurious eigenvalues are also *migrating*, i.e. they change from one iteration to another. An alternative strategy for stopping the Lanczos method is based on changing the initial vector $\boldsymbol{v}_1$. As the spurious eigenvalues are generated by the rounding errors, they will be generally different varying the source of such errors. Both changing the initial vector and checking the stability of the eigenvalues of T_m with m increasing allows for a relatively easy recognition of the right eigenvalues of A in practice.

3.3 Krylov subspace methods

The Lanczos method described above possesses a practical limitation quite similar to the CG algorithm for symmetric linear systems. Like CG, also the Lanczos algorithm should theoretically stop after n steps providing the desired solution, but because of the rounding errors it is necessary to proceed the iterations for $m > n$ steps. We observe on passing that the reason is exactly the same for the two algorithms and relies in the quick loss of the theoretical properties of the three-term Lanczos recurrence which is the actual basis of CG, too. As with CG, using the Lanczos algorithm for a number of iterations even much larger than the native matrix size can easily become prohibitive when dealing with large values of n.

To cope with such difficulty, suppose to stop Algorithm 7 after $m < n$ steps. Collecting the m vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$ as in (85) provides:

$$AV_m = V_m T_m \tag{89}$$

where T_m is the tridiagonal matrix (87) with size m. If $\boldsymbol{y}$ is an eigenvector of T_m with $\tilde{\lambda}$ the associated eigenvalue, then:

$$V_m T_m \boldsymbol{y} = \tilde{\lambda} V_m \boldsymbol{y} = A V_m \boldsymbol{y} \tag{90}$$

The last equality shows that any eigenvalue $\tilde{\lambda}$ of T_m is also an eigenvalue of A with $V_m \boldsymbol{y}$ the associated eigenvector. Hence the eigenvalues of T_m with $m < n$ are actually a subset of the eigenvalues of A. Their computation for relatively small values of m is quite cheap and provides immediately information on the eigenspectrum of A. In particular, the experience shows that the eigenvalues of T_m converge quickly to the extremal eigenvalues of A. In many applications, especially related to sparse large size matrices, the knowledge of the extremal eigenvalues is more important than that of the full spectrum. Algorithm 7 can be therefore conveniently changed as follows:

ALGORITHM 8: THE LANCZOS METHOD FOR EXTREMAL EIGENVALUES

1. Choose $\boldsymbol{v}_1$ such that $\|\boldsymbol{v}_1\|_2 = 1$
2. Set $\beta_1 = 1$ and $\boldsymbol{v}_0 = 0$
3. $\alpha_1 = \boldsymbol{v}_1^T A \boldsymbol{v}_1$
4. DO $m = 2, \ldots$ until convergence
5. $\quad \hat{\boldsymbol{v}}_m = A\boldsymbol{v}_{m-1} - \alpha_{m-1}\boldsymbol{v}_{m-1} - \beta_{m-1}\boldsymbol{v}_{m-2}$
6. $\quad \beta_m = \|\hat{\boldsymbol{v}}_m\|_2$
7. $\quad \boldsymbol{v}_m = \hat{\boldsymbol{v}}_m / \beta_m$
8. $\quad \alpha_m = \boldsymbol{v}_m^T A \boldsymbol{v}_m$
9. $\quad$ Update T_m with β_m and α_m
10. $\quad$ Use the QR method with T_m
11. END DO

Convergence of Algorithm 8 is achieved when the desired number of extremal eigenvalues are stable during the iterations. If a small number of extremal eigenvalues is required, the convergence of Algorithm 8 can be very fast.

Let us see the Lanczos method from another point of view. The vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$ computed after m steps of Algorithm 8 are an orthonormal basis of the Krylov subspace:

$$\mathcal{K} = \text{span}\left\{\boldsymbol{v}_1, A\boldsymbol{v}_1, A^2\boldsymbol{v}_1, \ldots, A^{m-1}\boldsymbol{v}_1\right\} \tag{91}$$

Such an observation follows immediately from the fact that any $\boldsymbol{v}_i$ can be actually written as the product of an (i-1)-degree polynomial in A by $\boldsymbol{v}_1$. The eigenvector of A:

$$\tilde{\boldsymbol{u}} = V_m \boldsymbol{y} \tag{92}$$

associated to the eigenvalue $\tilde{\lambda}$ of T_m (equation (90)) can be viewed as an approximation of the exact eigenvector $\boldsymbol{u}$ in $\mathcal{K}$. To compute the approximant $\tilde{\boldsymbol{u}}$ we need m independent constraints allowing for the univocal determination of $\boldsymbol{y}$. Following the Petrov-Galerkin mathematical framework, $\boldsymbol{y}$ can be computed by imposing m independent orthogonality constraints. If we prescribe that the residual of problem (1) with the approximant $\tilde{\boldsymbol{u}}$ is orthogonal to the m independent vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$, we obtain:

$$\boldsymbol{v}_i^T \left(A\tilde{\boldsymbol{u}} - \tilde{\lambda}\tilde{\boldsymbol{u}}\right) = 0 \qquad i = 1, \ldots, m \tag{93}$$

Using equation (92) into (93) and recalling the definition of V_m provides:

$$V_m^T A V_m \boldsymbol{y} = \tilde{\lambda}\boldsymbol{y} \tag{94}$$

which is exactly equivalent to equation (90). Therefore, Algorithm 8 can be interpreted as the orthogonal projection of the native eigenproblem (1) into Krylov subspaces of increasing size. The

pairs $(\tilde{\lambda}, \tilde{\boldsymbol{u}})$ solution of the projected eigenproblem quickly converge the the extremal pairs $(\lambda, \boldsymbol{u})$ solution of the native eigenproblem. The tridiagonal matrix T_m is also denoted as the orthogonal projection of A into $\mathcal{K}$.

The above framework can now be easily extended to non symmetric matrices. Recall that the recurrence consisting of equations (78), (80) and (81) still works for generating an orthonormal basis for $\mathcal{K}$ even with non symmetric matrices. By distinction, equation (82) is no longer true, with the coefficients h_{jk} generally non zero for any $j \leq k$. Moreover, equation (83) with a non symmetric matrix shows that $\|\hat{\boldsymbol{v}}_{k+1}\|_2$ is actually $h_{k+1,k}$. As a consequence, all the coefficients h_{jk} form a matrix H_m of order m with a Hessemberg form, i.e.:

$$V_m^T A V_m = H_m = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \cdots & h_{1,m} \\ h_{2,1} & h_{2,2} & h_{2,3} & \cdots & h_{2,m} \\ & h_{3,2} & h_{3,3} & \cdots & h_{3,m} \\ & & \ddots & & \vdots \\ & & & h_{m,m-1} & h_{m,m} \end{bmatrix} \tag{95}$$

The matrix H_m is the new orthogonal projection of A over $\mathcal{K}$. The resulting algorithm is due to Arnoldi:

ALGORITHM 9: THE ARNOLDI METHOD FOR NON SYMMETRIC MATRICES

1. Choose $\boldsymbol{v}_1$ such that $\|\boldsymbol{v}_1\|_2 = 1$
2. $h_{1,1} = \boldsymbol{v}_1^T A \boldsymbol{v}_1$
3. DO $m = 2, \ldots$ until convergence
4. $\quad \hat{\boldsymbol{v}}_m = A\boldsymbol{v}_{m-1} - \sum_{j=1}^{m-1} h_{j,m-1}\boldsymbol{v}_j$
5. $\quad h_{m,m-1} = \|\hat{\boldsymbol{v}}_m\|_2$
6. $\quad \boldsymbol{v}_m = \hat{\boldsymbol{v}}_m / h_{m,m-1}$
7. $\quad$ DO $j = 1, \ldots, m$
8. $\quad\quad h_{j,m} = \boldsymbol{v}_m^T A \boldsymbol{v}_j$
9. $\quad$ END DO
10. $\quad$ Update H_m with $h_{m,m-1}$ and $h_{1,m}, h_{2,m}, \ldots, h_{m,m}$
11. $\quad$ Use the QR method with H_m
12. END DO

The Arnoldi method is much more expensive than Lanczos because we have to store all vectors $\boldsymbol{v}_i$ and a full Hessemberg matrix instead of a tridiagonal one. Moreover, the QR method used with H_m is more expensive. In other words, the Arnoldi method is a long-term recurrence. It should be interesting developing a short-term recurrence similar to Algorithm 8 also for the non symmetric case. An alternative technique relies on replacing the orthogonal projection with a non orthogonal

one, i.e. the residual of eigenproblem (1) projected on $\mathcal{K}$ is orthogonalized with respect to another Krylov subspace $\mathcal{L}$. Hence, a second basis $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m$ for $\mathcal{L}$ is needed. The most convenient choice relies on setting:

$$\mathcal{L} = \text{span}\left\{\boldsymbol{w}_1, A^T\boldsymbol{w}_1, \left(A^T\right)^2\boldsymbol{w}_1, \ldots, \left(A^T\right)^{m-1}\boldsymbol{w}_1\right\} \tag{96}$$

i.e. $\mathcal{L}$ is the Krylov subspace associated with A^T. Now it is possible to generate two bases $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$ and $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m$ for $\mathcal{K}$ and $\mathcal{L}$, respectively, which are mutually bi-orthogonal. Writing the recurrence (78) for vectors $\boldsymbol{w}_i$:

$$\hat{\boldsymbol{w}}_{k+1} = A^T\boldsymbol{w}_k - \sum_{j=1}^{k}\tilde{h}_{jk}\boldsymbol{w}_j \tag{97}$$

and prescribing the bi-orthogonality condition with vectors $\boldsymbol{v}_i$ gives:

$$h_{jk} = \tilde{h}_{kj} = \boldsymbol{w}_j^T A\boldsymbol{v}_k \qquad j = 1, \ldots, k \tag{98}$$

As in the Arnoldi algorithm, the coefficients $\tilde{h}_{jk}$ belong to a matrix with the Hessemberg form. Denoting as $\tilde{H}$ such matrix, from equation (98) it follows that:

$$H = \tilde{H}^T = \tilde{T} \tag{99}$$

Equation (99) actually means that $\tilde{T}$ must be tridiagonal. The vectors $\hat{\boldsymbol{v}}_{k+1}$ and $\hat{\boldsymbol{w}}_{k+1}$ now are to be scaled so that their inner product is unitary. Following the idea of equation (83), it can be easily seen that the following relationships:

$$\boldsymbol{v}_{k+1} = \frac{\hat{\boldsymbol{v}}_{k+1}}{\tilde{h}_{k,k+1}} \tag{100}$$

$$\boldsymbol{w}_{k+1} = \frac{\hat{\boldsymbol{w}}_{k+1}}{h_{k,k+1}} \tag{101}$$

allow for satisfying the above requirement.

The non zero coefficients of H and $\tilde{H}$ are classically denoted as:

$$h_{k,k} = \tilde{h}_{k,k} = \alpha_k \tag{102}$$

$$h_{k-1,k} = \tilde{h}_{k,k-1} = \beta_k \tag{103}$$

$$h_{k,k-1} = \tilde{h}_{k-1,k} = \delta_k \tag{104}$$

The tridiagonal matrix $\tilde{T}_m$ built at the m-th step of the algorithm above is therefore:

$$W_m^T A V_m = \tilde{T}_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & & \cdots & & \\ & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \delta_m & \alpha_m \end{bmatrix} \tag{105}$$

with W_m the matrix whose columns are the vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m$. $\tilde{T}_m$ is nothing but the projection of A onto $\mathcal{K}$ orthogonally to $\mathcal{L}$. Putting the relationships above together with the reasoning underlying Algorithm 8 and Algorithm 9 finally provides the following algorithm:

ALGORITHM 10: THE TWO-SIDED LANCZOS METHOD FOR NON SYMMETRIC MATRICES

1. Choose $\boldsymbol{v}_1$ and $\boldsymbol{w}_1$ such that $\boldsymbol{w}_1^T \boldsymbol{v}_1 = 1$
2. Set $\beta_1 = \delta_1 = 0$ and $\boldsymbol{v}_0 = \boldsymbol{w}_0 = 0$
3. $\alpha_1 = \boldsymbol{w}_1^T A \boldsymbol{v}_1$
4. DO $m = 2, \ldots$ until convergence
5. $\quad \hat{\boldsymbol{v}}_m = A\boldsymbol{v}_{m-1} - \alpha_{m-1}\boldsymbol{v}_{m-1} - \beta_{m-1}\boldsymbol{v}_{m-2}$
6. $\quad \hat{\boldsymbol{w}}_m = A^T\boldsymbol{w}_{m-1} - \alpha_{m-1}\boldsymbol{w}_{m-1} - \delta_{m-1}\boldsymbol{w}_{m-2}$
7. $\quad$ Choose β_m and δ_m such that $\beta_m\delta_m = \hat{\boldsymbol{w}}_m^T\hat{\boldsymbol{v}}_m$
8. $\quad \boldsymbol{v}_m = \hat{\boldsymbol{v}}_m/\delta_m$
9. $\quad \boldsymbol{w}_m = \hat{\boldsymbol{w}}_m/\beta_m$
10. $\quad \alpha_m = \boldsymbol{w}_m^T A \boldsymbol{v}_m$
11. $\quad$ Update $\tilde{T}_m$ with α_m, β_m and δ_m
12. $\quad$ Use the QR method with $\tilde{T}_m$
13. END DO

Theoretically, there are infinite ways of choosing β_m and δ_m so as to accomplish the condition in step 7 of Algorithm 10. The most usual technique is the following:

$$\delta_m = |\hat{\boldsymbol{w}}_m^T\hat{\boldsymbol{v}}_m|^{1/2} \tag{106}$$

$$\beta_m = \hat{\boldsymbol{w}}_m^T\hat{\boldsymbol{v}}_m/\delta_m \tag{107}$$

A consequence of equations (106) and (107) is that the coefficients δ_m are always positive with $\beta_m = \pm\delta_m$.

It can be easily seen that Algorithm 10 appears as a somewhat natural extension for the non symmetric case of the original Lanczos method (Algorithm 8). The main appeal with respect to the Arnoldi method is that an elegant and cheap short-term recurrence is restored. However, there are more chances for the algorithm to break down. Algorithm 10 will abort whenever $\hat{\boldsymbol{w}}_m^T\hat{\boldsymbol{v}}_m$ is null, i.e. either $\hat{\boldsymbol{v}}_m$ or $\hat{\boldsymbol{w}}_m$ is the zero vector, or their inner product is zero even though $\hat{\boldsymbol{v}}_m$ and $\hat{\boldsymbol{w}}_m$ are both nonzero. The case $\hat{\boldsymbol{v}}_m = 0$ is called *lucky breakdown* because it means that the subspace $\mathcal{K}$ is invariant with A, i.e. the projection $\tilde{T}_{m-1}$ coincides with A, and all the eigenvalues have already been computed. The case $\hat{\boldsymbol{w}}_m = 0$ means that the subspace $\mathcal{L}$ is invariant with A^T and unfortunately nothing can be said for the native eigenproblem. The case $\hat{\boldsymbol{w}}_m^T\hat{\boldsymbol{v}}_m = 0$ with both $\hat{\boldsymbol{v}}_m$ and $\hat{\boldsymbol{w}}_m$ nonzero is called *serious breakdown* and again nothing can be said for the native

eigenproblem. This difficulty can be cured using special *Look-ahead Lanczos strategies* which allow for computing the pair $\hat{\boldsymbol{v}}_{m+1}$ and $\hat{\boldsymbol{w}}_{m+1}$ even if $\boldsymbol{v}_m$ and $\boldsymbol{w}_m$ are not defined. Unfortunately, the use of look-ahead strategies causes the loss of the short-term recurrences in Algorithm 10 with a consequent larger storage requirement. Recall that in computer arithmetic near breakdowns occurring when the inner product $\hat{\boldsymbol{w}}_m^T \hat{\boldsymbol{v}}_m$ is small are potentially as dangerous as real breakdowns.