Iterative methods for sparse linear systems

Massimiliano Ferronato
Dept. Civil, Environmental and Architectural Engineering (ICEA)
University of Padova

E-mail: massimiliano.ferronato@unipd.it
http://www.dmsa.unipd.it/~ferronat

Contents

1 Basic concepts on iterative methods

Consider the linear system:

$$A\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

with A an $(n \times n)$ real matrix and $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^n$. If A is non singular the unique solution to (1) is:

$$\boldsymbol{h} = A^{-1}\boldsymbol{b} \tag{2}$$

However, the explicit inversion of a matrix A arising from typical engineering applications, such as the discretization with Finite Elements or Finite Differences of Partial Differential Equations (PDEs), is a task that cannot be realistically undertaken even on current supercomputers. The main reason is twofold. First, A is typically large ($n \geq 10^6$) but very sparse, however A^{-1} is in general structurally dense and cannot be entirely stored. Second, the computation of A^{-1} has a cost on the order of n^3 operations, that would mean years also on modern CPUs. This is why several algorithms have been developed to solve numerically the system (1) avoiding the explicit computation of A^{-1}.

Though in the last decade there has been an increasing contamination of ideas, for the sake of simplicity we can still subdivide linear system solvers into two big classical groups, i.e., direct and iterative methods. Roughly speaking, direct methods provide $\boldsymbol{h}$ through a sequence of elementary operations whose number depends on the size n of matrix A. The most common direct solvers are based on Gaussian elimination accelerated by proper scaling and reordering algorithms. For large and sparse matrices arising from 3D problems, direct methods are rarely the most efficient choice, as the required CPU time and storage grow very quickly with n. Just to give an idea, to solve a relatively small linear system with $n = 21,355$ and 4,425,300 non-zero coefficients in A, requiring approximately 60 kbyte of storage, the total memory needed by a state-of-the-art Harwell Software Library (HSL) routine is about 2 Gbyte. On the other hand, iterative methods find $\boldsymbol{h}$ by building a converging sequence of approximants $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., $\boldsymbol{x}_k$, starting from an initial guess $\boldsymbol{x}_0$. Quite obviously, the method is *convergent* if:

$$\lim_{k\to\infty} \boldsymbol{x}_k = \boldsymbol{h} \tag{3}$$

Defining the error vector $\boldsymbol{e}_k$ as:

$$\boldsymbol{e}_k = \boldsymbol{h} - \boldsymbol{x}_k \tag{4}$$

the convergence condition (3) is equivalent to:

$$\lim_{k\to\infty} \|\boldsymbol{e}_k\| = 0 \tag{5}$$

The construction of the next approximant $\boldsymbol{x}_{k+1}$ is obtained from $\boldsymbol{x}_k$ by a recurrent relationship that typically requires only matrix-vector and scalar products.

A convergent iterative method produces theoretically the exact solution $\boldsymbol{h}$ after an infinite number of steps. However, the procedure can be stopped when the approximate solution $\boldsymbol{x}_k$ is accurate enough. If this occurs after a small number of iterations, then

iterative methods can be much more efficient than direct methods. The accuracy of the current approximate solution can be evaluated using the *relative error*:

$$e_r = \frac{\|\boldsymbol{e}_k\|}{\|\boldsymbol{h}\|} \tag{6}$$

As $\boldsymbol{h}$ is not known, the relative error e_r cannot be used in practice as exit test. Let us define the *residual* vector $\boldsymbol{r}_k$ as:

$$\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k \tag{7}$$

It is easy to see that:

$$\boldsymbol{r}_k = A\boldsymbol{h} - A\boldsymbol{x}_k = A\boldsymbol{e}_k \tag{8}$$

hence in a convergent iterative method the following relation holds true:

$$\lim_{k\to\infty} \|\boldsymbol{r}_k\| = 0 \tag{9}$$

The accuracy of the current approximate solution can be therefore evaluated using also the *relative residual* r_r as exit test:

$$r_r = \frac{\|\boldsymbol{r}_k\|}{\|\boldsymbol{b}\|} \tag{10}$$

The exit test on r_r is much more practical than that on e_r because the residual vector can be always computed easily. However, it is important to link e_r to r_r in order to ensure its reliability. Write $\boldsymbol{e}_k$ from equation (8) and use the well-known properties of compatible matrix norms:

$$\|\boldsymbol{e}_k\| \leq \left\|A^{-1}\right\| \ \|\boldsymbol{r}_k\| \tag{11}$$

Similarly, recalling that $\boldsymbol{b} = A\boldsymbol{h}$ we have:

$$\|\boldsymbol{b}\| \leq \|A\| \ \|\boldsymbol{h}\| \tag{12}$$

Combining the inequality (11) with (12), it can be easily obtained that:

$$\frac{\|\boldsymbol{e}_k\|}{\|\boldsymbol{h}\|} \leq \|A\| \left\|A^{-1}\right\| \frac{\|\boldsymbol{r}_k\|}{\|\boldsymbol{b}\|} \tag{13}$$

Defining $\kappa(A) = \|A\| \ \|A^{-1}\|$ as the *conditioning number* of A, inequality (13) yields:

$$e_r \leq \kappa\left(A\right) r_r \tag{14}$$

Therefore, a small relative residual does not guarantee that the relative error is also small if the conditioning number of A is large. A better exit test should be:

$$\kappa\left(A\right) r_r \leq \ \text{TOL} \tag{15}$$

but the cost and difficulty in computing $\kappa(A)$ can prevent its use. If A is symmetric and positive definite and the 2-norm is used in the definition of r_r and e_r, the conditioning number $\kappa(A)$ turns out to be:

$$\kappa\left(A\right) = \frac{\lambda_1\left(A\right)}{\lambda_n\left(A\right)} \tag{16}$$

where $\lambda_1(A)$ and $\lambda_n(A)$ are the largest and smallest eigenvalue of A, respectively. The value of $\kappa(A)$ computed as in (16) is also called *spectral conditioning number* of A. Hence, for symmetric positive definite matrices the eigenspectrum of A can provide a measure of the conditioning of the problem and the quality of the exit test on the relative residual.

The most classical iterative methods are stationary algorithms, such as Richardson, Jacobi, Seidel and Successive Overrelation. These techniques converge linearly to the solution $\boldsymbol{h}$ provided that the the spectral radius of the iteration matrix E:

$$E = I - M^{-1}A \tag{17}$$

is smaller than 1. In equation (17), M^{-1} is defined as a *preconditioner* for A, in the sense that the product $M^{-1}A$ should be as close to the identity as possible. Stationary methods are not competitive with modern direct methods because of their slow convergence rate. More efficient techniques are based on the Krylov subspaces generated by matrix A. In particular, special variants of these methods can be obtained if A is symmetric positive definite.

2 Gradient methods for symmetric positive definite systems

If A is symmetric and positive definite, the solution to the system (1) can be easily recasted as an optimization problem. Generally speaking, an optimization problem consists of finding the minimum of a cost function Φ depending on a vector of n parameters $\boldsymbol{x}$ within an admittable domain $\Omega \subseteq \mathbb{R}^n$. The solution to an optimization problem is the combination of parameters providing the minimum cost Φ. The *gradient methods* represent a class of techniques used to solve optimization problems.

There are several ways to find the minimum of Φ. If Φ is derivable and its derivatives can be easily computed, the direction of the *gradient* of Φ starting from an initial guess $\boldsymbol{x}_0$ can be used to search efficiently the minimum. Let us define as gradient of Φ the vector $\nabla\Phi$:

$$\nabla\Phi = \left[\frac{\partial\Phi}{\partial x_1}, \frac{\partial\Phi}{\partial x_2}, \ldots, \frac{\partial\Phi}{\partial x_n}\right]^T \tag{18}$$

Given an oriented direction $\boldsymbol{s} \in \mathbb{R}^n$, the function $\Phi(\boldsymbol{x})$ either increases or decreases along $\boldsymbol{s}$ in the neighbourhood of $\boldsymbol{x}_0$ if the scalar product $\boldsymbol{s}^T\nabla\Phi(\boldsymbol{x}_0)$ is either positive or negative, respectively. Hence, the steepest variation of Φ in the neighbourhood of $\boldsymbol{x}_0$ can be found just along the direction of $\nabla\Phi(\boldsymbol{x}_0)$. The gradient methods are iterative techniques that build a sequence of approximations of the minimum of Φ moving at the k-th step along $-\nabla\Phi(\boldsymbol{x}_k)$ and defining $\boldsymbol{x}_{k+1}$ as the local minimum of Φ along that direction. This way an n-dimensional problem is reduced to a sequence of one-dimensional minimum problems.

2.1 The Steepest Descent (SD)

Let us define the cost function $\Phi : \mathbb{R}^n \to \mathbb{R}$ as:

$$\Phi(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{h})^T A(\boldsymbol{x} - \boldsymbol{h}) \tag{19}$$

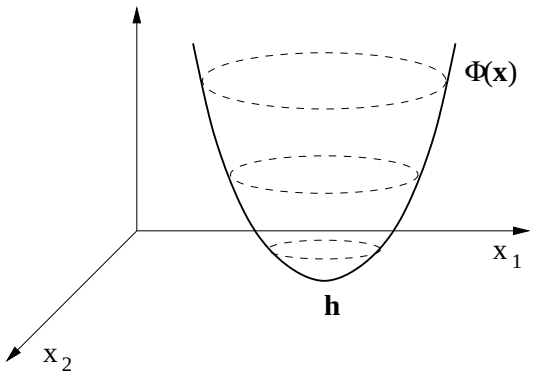

Figure 1: Cost function $\Phi(\boldsymbol{x})$ for $n = 2$.

Recalling the error definition (4), equation (19) can be also written as:

$$\Phi(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{e}^T A \boldsymbol{e} \tag{20}$$

If A is symmetric and positive definite, Φ is certainly positive for any $\boldsymbol{e} \neq 0$ and is minimum for $\boldsymbol{e} = 0$, i.e., $\boldsymbol{x} = \boldsymbol{h}$. The cost function $\Phi(\boldsymbol{x})$ defined in (19) is a quadratic form that has its unique and absolute minimum in $\boldsymbol{h}$. Hence, the solution $\boldsymbol{h}$ can be computed by minimizing $\Phi(\boldsymbol{x})$.

It is possible to reconstruct the shape of the cost function by analyzing its *equipotential surfaces*:

$$\Phi(\boldsymbol{x}) = c \tag{21}$$

where c is a constant real parameter. Equation (21) defines a class of surfaces depeding on c. To better understand the structure of these surfaces, let us operate a coordinate change using the eigenvectors $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, ..., $\boldsymbol{u}_n$ as new reference system. As A is symmetric positive definite, $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, ..., $\boldsymbol{u}_n$ generate an orthonormal basis of $\mathbb{R}^n$. The error $\boldsymbol{e}$ in the new reference reads:

$$\boldsymbol{e} = z_1\boldsymbol{u}_1 + z_2\boldsymbol{u}_2 + \ldots + z_n\boldsymbol{u}_n \tag{22}$$

where z_i is the component of $\boldsymbol{e}$ along the eigenvector u_i. Collecting the z_i values in the vector $\boldsymbol{z}$ and the eigenvectors $\boldsymbol{u}_i$ as columns of the matrix U, equation (22) can be written in a compact form as:

$$\boldsymbol{e} = U\boldsymbol{z} \tag{23}$$

Recall that U is orthogonal, i.e., $U^T = U^{-1}$. Moreover, A is also diagonalizable. Denoting by Λ the diagonal matrix containing its eigenvalues, A can be written as:

$$A = U\Lambda U^T \tag{24}$$

Using equations (23) and (24), the equipotential surface (21) in the new reference of the eigenvectors of A reads:

$$\Phi(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T U^T U \Lambda U U^T \boldsymbol{z} = \frac{1}{2}\boldsymbol{z}^T \Lambda \boldsymbol{z} = c \tag{25}$$

i.e.:

$$\lambda_1 z_1^2 + \lambda_2 z_2^2 + \ldots + \lambda_n z_n^2 = 2c \tag{26}$$

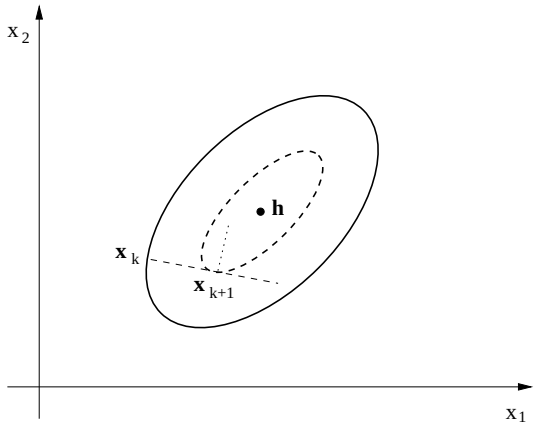

Figure 2: New approximation $\boldsymbol{x}_{k+1}$ for $n = 2$.

The geometrical interpretation of equation (26) is trivial for $n = 2$. In fact, it reads:

$$\frac{z_1^2}{2c/\lambda_1} + \frac{z_2^2}{2c/\lambda_2} = 1 \tag{27}$$

that is the equation of an ellipse referred to its axes. Hence, $\Phi(\boldsymbol{x})$ with $n = 2$ is nothing but a positive paraboloid with vertex in $\boldsymbol{h}$ (Figure 1). The equipotential surfaces consist of a set of concentric ellipsis whose axes have the direction of the eigenvectors of A and length equal to:

$$2\sqrt{\frac{2c}{\lambda_1}}, \qquad 2\sqrt{\frac{2c}{\lambda_2}} \tag{28}$$

The solution $\boldsymbol{h}$ is the center of the set of ellipsis. In $\mathbb{R}^n$, the equipotential surfaces of $\Phi(\boldsymbol{x})$ are a set of hyperellipsoids with axes defined by the eigenvectors of A and length depending on the eigenvalues of A according to a generalization of equation (28). The geometric center of this set of hyperellipsoids is the solution $\boldsymbol{h}$.

The basic idea is to reach the minimum of Φ, i.e., the center of the set of hyperellipsoids associated to A, moving along the direction of the gradient of Φ. Assume that at the k-th step we know the tentative solution $\boldsymbol{x}_k$ and we want to find a new approximation $\boldsymbol{x}_{k+1}$ closer to $\boldsymbol{h}$. The gradient of Φ in $\boldsymbol{x}_k$ reads:

$$\nabla\Phi\left(\boldsymbol{x}_k\right) = \frac{1}{2}\nabla\left[\left(\boldsymbol{x}_k - \boldsymbol{h}\right)^T A\left(\boldsymbol{x}_k - \boldsymbol{h}\right)\right] \tag{29}$$

Recalling that $A\boldsymbol{h} = \boldsymbol{b}$ and $A = A^T$, equation (29) yields:

$$\nabla\Phi\left(\boldsymbol{x}_k\right) = \frac{1}{2}\nabla\left[\boldsymbol{x}_k^T A\boldsymbol{x}_k - 2\boldsymbol{x}_k^T\boldsymbol{b} + \boldsymbol{h}^T\boldsymbol{b}\right] \tag{30}$$

Deriving with respect to $\boldsymbol{x}_k$ we obtain:

$$\nabla\Phi\left(\boldsymbol{x}_k\right) = A\boldsymbol{x}_k - \boldsymbol{b} = -\boldsymbol{r}_k \tag{31}$$

i.e., the direction of the gradient of the cost function in $\boldsymbol{x}_k$ is equal to that of the residual vector $\boldsymbol{r}_k$. As the gradient is oriented toward an increase of $\Phi(\boldsymbol{x})$, to find the minimum

we should move in the opposite way, i.e., along $-\nabla\Phi(\boldsymbol{x}_k) = \boldsymbol{r}_k$. By definition of gradient, the residual vector $\boldsymbol{r}_k$ is also orthogonal to the equipotential surface obtained setting $c = \Phi(\boldsymbol{x}_k)$. Starting from $\boldsymbol{x}_k$ and moving along $\boldsymbol{r}_k$, we define $\boldsymbol{x}_{k+1}$ as the point where $\Phi(\boldsymbol{x})$ is locally minimum (Figure 2). This is an easy problem, as it coincides with the minimization of a parabola. The new approximation $\boldsymbol{x}_{k+1}$ can be written as:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k \tag{32}$$

where α_k is the real parameter such that $\Phi(\boldsymbol{x}_{k+1})$ is minimum:

$$\alpha_k = \text{argmin}_t \Phi\left(\boldsymbol{x}_k + t\boldsymbol{r}_k\right) \tag{33}$$

Such a minimum is uniquely determined by deriving $\Phi(\boldsymbol{x}_{k+1})$ with respect to α_k and setting to 0. Using the chain rule to differentiate yields:

$$\frac{\partial\Phi}{\partial\alpha_k} = \frac{\partial\Phi}{\partial\boldsymbol{x}_{k+1}} \cdot \frac{\partial\boldsymbol{x}_{k+1}}{\partial\alpha_k} = \boldsymbol{r}_k^T \left(A\boldsymbol{x}_{k+1} - \boldsymbol{b}\right) = 0 \tag{34}$$

where, recalling equation (32), we obtain:

$$\boldsymbol{r}_k^T \left(-\boldsymbol{r}_k + \alpha_k A\boldsymbol{r}_k\right) = 0 \qquad \Rightarrow \qquad \alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{r}_k^T A\boldsymbol{r}_k} \tag{35}$$

Equation (34) suggests two important observations. First, the minimum condition for Φ along $\boldsymbol{r}_k$ yields:

$$\boldsymbol{r}_k^T \boldsymbol{r}_{k+1} = 0 \tag{36}$$

i.e., consecutive residual vectors are orthogonal. Second, comparing equations (34) and (35) we obtain:

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{r}_k \tag{37}$$

i.e., it is possible to compute the new residual making no use of $\boldsymbol{x}_{k+1}$. The recurrent relationship (37) is more convenient than the definition of residual vector from a computational point of view, as it uses again the product $A\boldsymbol{r}_k$, already stored for α_k, instead of computing also $A\boldsymbol{x}_{k+1}$.

The resulting iterative method is called *Steepest Descent* (SD) and can be summarized by the following algorithm:

Algorithm 1: Steepest Descent

1. Choose $\boldsymbol{x}_0$
2. Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
3. Do $k = 0, \ldots$ until convergence
4. $\quad \alpha_k = \boldsymbol{r}_k^T \boldsymbol{r}_k / \boldsymbol{r}_k^T A\boldsymbol{r}_k$
5. $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k$
6. $\quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{r}_k$
7. End Do

The convergence rate of the SD method is governed by:

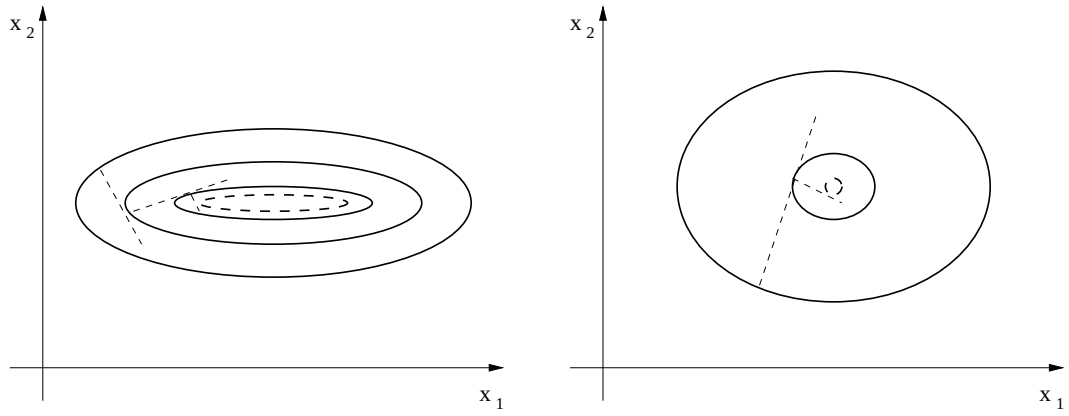

Figure 3: Convergence of the SD method for $n = 2$.

THEOREM. *Let A be symmetric and positive definite. Then the energy A-norm of $\boldsymbol{e}_k$ generated by Algorithm 1 satisfy the inequality:*

$$\|\boldsymbol{e}_k\|_A \leq \mu^k \|\boldsymbol{e}_0\|_A \tag{38}$$

with $\mu = \frac{\kappa(A)-1}{\kappa(A)+1}$, and Algorithm 1 converges for any initial guess $\boldsymbol{x}_0$.

PROOF. The energy A-norm of a vector $\boldsymbol{d}$ is defined as:

$$\|\boldsymbol{d}\|_A = \sqrt{\boldsymbol{d}^T A \boldsymbol{d}} \tag{39}$$

The result (38) follows immediately applying the Kantorovich inequality. For details, see Saad (2003), *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, pp. 138–140. □

The previous theorem guarantees the convergence of the SD method and provides an upper bound for the error decrease:

$$\frac{\|\boldsymbol{e}_k\|_A}{\|\boldsymbol{e}_0\|_A} \leq \mu^k \tag{40}$$

For k going to infinity, the right-hand side of (40) goes to 0 as $\mu < 1$. In particular, if $\lambda_1 = \lambda_2 = \ldots = \lambda_n$ then $\mu = 0$ and Algorithm 1 converges to $\boldsymbol{h}$ in just one iteration for any initial guess $\boldsymbol{x}_0$. However, if $\kappa(A)$ is much larger than 1, then $\mu \simeq 1$ and convergence can be very slow. Unfortunately, this is typically the case with matrices arising from the discretization of PDEs. An intuitive reason for such a behavior is schematically provided in Figure 3 using the equipotential surfaces of a system with $n = 2$. The search directions generate a zig-zag path toward the geometric center of the ellipses. With eccentric ellipses the number of iterations required to achieve the convergence can be quite large, while with "almost circular" ellipses convergence is fast independently of the starting point.

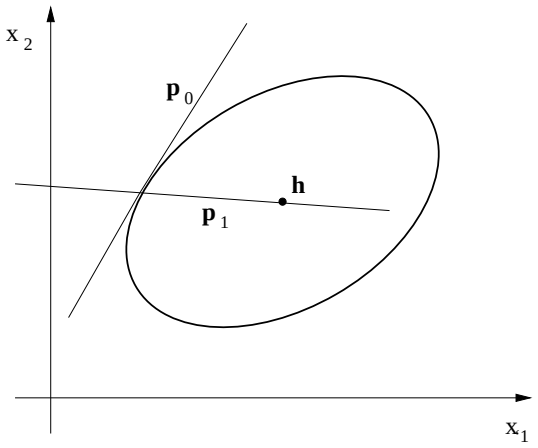

Figure 4: Pair of conjugate directions with respect to an ellipse.

2.2 The Conjugate Gradient (CG)

The Conjugate Gradient (CG) method was introduced in 1952 by Hestenes and Stiefel as an acceleration of the SD algorithm. The basic idea consists of modifying the search direction for the computation of $\boldsymbol{x}_{k+1}$ so as to produce a faster convergence to the global minimum of Φ.

Given $\boldsymbol{x}_k$, the new approximation $\boldsymbol{x}_{k+1}$ is computed with a recurrent relationship similar to (32) where a new search direction $\boldsymbol{p}_k$ is used instead of $\boldsymbol{r}_k$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \tag{41}$$

The initial search direction $\boldsymbol{p}_0$ is set to $\boldsymbol{r}_0$ as in the SD scheme. The new search direction $\boldsymbol{p}_1$ is computed as a correction to the gradient of Φ in $\boldsymbol{x}_1$:

$$\boldsymbol{p}_1 = \boldsymbol{r}_1 + \beta_0 \boldsymbol{p}_0 \tag{42}$$

such that $\boldsymbol{p}_1$ be A-orthogonal to $\boldsymbol{p}_0$, i.e.:

$$\boldsymbol{p}_1^T A \boldsymbol{p}_0 = 0 \tag{43}$$

Introducing equation (42) into (43) allows for computing β_0 as:

$$\beta_0 = -\frac{\boldsymbol{r}_1^T A \boldsymbol{p}_0}{\boldsymbol{p}_0^T A \boldsymbol{p}_0} \tag{44}$$

Geometrically, two A-orthogonal directions are *conjugate* with respect to the conic generated by A. For example, Figure 4 shows a pair of conjugate directions with respect to the conic generated by a 2×2 symmetric positive definite matrix, i.e., an ellipse. If $\boldsymbol{p}_0$ is tangential to the ellipse, then $\boldsymbol{p}_1$ goes towards the geometric center. Hence, it can be easily deduced that the CG method will converge to $\boldsymbol{h}$ in a 2×2 linear system in just two iterations.

Generalizing equations (42) and (44) to any step k yields:

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k \tag{45}$$

with:

$$\beta_k = -\frac{\boldsymbol{r}_{k+1}^T A \boldsymbol{p}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k} \tag{46}$$

The expression for the scalar α_k in (41) is slightly different than in the SD method because of the use of the new search direction $\boldsymbol{p}_k$. Recalling the chain rule to differentiate $\Phi(\boldsymbol{x}_{k+1})$ and using equation (41) provides:

$$\frac{\partial \Phi}{\partial \alpha_k} = \frac{\partial \Phi}{\partial \boldsymbol{x}_{k+1}} \cdot \frac{\partial \boldsymbol{x}_{k+1}}{\partial \alpha_k} = \boldsymbol{p}_k^T \left(A \boldsymbol{x}_{k+1} - \boldsymbol{b} \right) = 0 \tag{47}$$

that gives:

$$\boldsymbol{p}_k^T \left(-\boldsymbol{r}_k + \alpha_k A \boldsymbol{p}_k \right) = 0 \qquad \Rightarrow \qquad \alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{p}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k} \tag{48}$$

Equation (48) shows also that $\boldsymbol{p}_k$ and $\boldsymbol{r}_{k+1}$ are orthogonal for any k, and that the residual vector at the $(k+1)$-th step can be also computed as:

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k \tag{49}$$

which is a convenient recurrent relationship similar to the one used in the SD scheme. The final CG method can be summarized as follows:

ALGORITHM 2: CONJUGATE GRADIENT

1. Choose $\boldsymbol{x}_0$
2. Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$ and set $\boldsymbol{p}_0 = \boldsymbol{r}_0$
3. DO $k = 0, \ldots$ until convergence
4. $\quad \alpha_k = \boldsymbol{r}_k^T \boldsymbol{p}_k / \boldsymbol{p}_k^T A \boldsymbol{p}_k$
5. $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
6. $\quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k$
7. $\quad \beta_k = -\boldsymbol{r}_{k+1}^T A \boldsymbol{p}_k / \boldsymbol{p}_k^T A \boldsymbol{p}_k$
8. $\quad \boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$
9. END DO

From a computational viewpoint, one CG iteration is only slightly more expensive than an SD iteration, implying one additional scalar product and vector update. However, the CG algorithm is much more robust than SD because of the following important result:

THEOREM. *Let A be symmetric positive definite, with $\boldsymbol{r}_0$, $\boldsymbol{r}_1$, ..., $\boldsymbol{r}_k$ and $\boldsymbol{p}_0$, $\boldsymbol{p}_1$, ..., $\boldsymbol{p}_k$ the sequence of residual vectors and search directions generated by k steps of Algorithm 2. Then the vectors $\boldsymbol{r}_{k+1}$ and $\boldsymbol{p}_{k+1}$ satisfy the equations:*

$$\boldsymbol{r}_{k+1}^T \boldsymbol{r}_j = 0 \qquad \forall\, j \leq k \tag{50}$$

$$\boldsymbol{p}_{k+1}^T A \boldsymbol{p}_j = 0 \qquad \forall\, j \leq k \tag{51}$$

PROOF. The theorem can be proved by induction. For $k = 0$ the thesis is trivially true. In fact, $\boldsymbol{p}_1^T A\boldsymbol{p}_0 = 0$ by definition, while using equation (49) with $\boldsymbol{p}_0 = \boldsymbol{r}_0$ gives:

$$\boldsymbol{r}_1^T \boldsymbol{r}_0 = \left(\boldsymbol{r}_0 - \frac{\boldsymbol{r}_0^T \boldsymbol{r}_0}{\boldsymbol{r}_0^T A\boldsymbol{r}_0} A\boldsymbol{r}_0 \right) \boldsymbol{r}_0 = 0 \tag{52}$$

Let us assume that the thesis holds true for k:

$$\boldsymbol{r}_k^T \boldsymbol{r}_j = 0 \qquad \forall\, j \leq k-1 \tag{53}$$

$$\boldsymbol{p}_k^T A\boldsymbol{p}_j = 0 \qquad \forall\, j \leq k-1 \tag{54}$$

and prove that equations (53) and (54) imply equations (50) and (51). Recalling (49), the condition (50) reads:

$$\boldsymbol{r}_{k+1}^T \boldsymbol{r}_j = (\boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k)^T \boldsymbol{r}_j = -\alpha_k \boldsymbol{p}_k^T A\boldsymbol{r}_j \tag{55}$$

Introduce in equation (55) the expression for $\boldsymbol{r}_j$ obtained from (45):

$$\boldsymbol{r}_{k+1}^T \boldsymbol{r}_j = -\alpha_k \left(\boldsymbol{p}_k^T A\boldsymbol{p}_j - \beta_{j-1} \boldsymbol{p}_k^T A\boldsymbol{p}_{j-1} \right) = 0 \tag{56}$$

Finally, using equation (45) into (51) yields:

$$\boldsymbol{p}_{k+1}^T A\boldsymbol{p}_j = (\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k)^T A\boldsymbol{p}_j = \boldsymbol{r}_{k+1}^T A\boldsymbol{p}_j \tag{57}$$

where, replacing the product $A\boldsymbol{p}_j$ obtained from (49), we have:

$$\boldsymbol{p}_{k+1}^T A\boldsymbol{p}_j = \frac{1}{\alpha_j} \left(\boldsymbol{r}_{k+1} \boldsymbol{r}_j - \boldsymbol{r}_{k+1} \boldsymbol{r}_{j-1} \right) = 0 \tag{58}$$

□

The previous result is also known as the *finite termination property* of CG. After n steps, the residual vector $\boldsymbol{r}_n$ is simultaneously orthogonal to all previous n residual vectors in $\mathbb{R}^n$. The only vector satisfying this condition is $\boldsymbol{r}_n = 0$, that implies $\boldsymbol{x}_n = \boldsymbol{h}$. Hence, the CG algorithm converges theoretically in at most n iterations.

An alternative formulation is also possible for Algorithm 2 exploiting a few useful relations. It is easy to prove that $\boldsymbol{p}_k^T \boldsymbol{r}_k = \boldsymbol{r}_k^T \boldsymbol{r}_k$. Recalling equation (45) we have:

$$\boldsymbol{p}_k^T \boldsymbol{r}_k = \boldsymbol{r}_k^T \boldsymbol{r}_k + \beta_{k-1} \boldsymbol{p}_{k-1}^T \boldsymbol{r}_k \tag{59}$$

The second addendum at the right-hand side of equation (59) can be also developed using the recurrent relation (45). As the residual vectors are orthogonal, equation (59) becomes:

$$\boldsymbol{p}_k^T \boldsymbol{r}_k = \boldsymbol{r}_k^T \boldsymbol{r}_k + \beta_{k-1} \beta_{k-2} \boldsymbol{p}_{k-2}^T \boldsymbol{r}_k \tag{60}$$

Replacing again $\boldsymbol{p}_{k-2}$ with the recurrent relation (45) and going backward until $\boldsymbol{p}_0$ we obtain:

$$\boldsymbol{p}_k^T \boldsymbol{r}_k = \boldsymbol{r}_k^T \boldsymbol{r}_k + \left(\prod_{j=0}^{k-1} \beta_j \right) \boldsymbol{r}_0^T \boldsymbol{r}_k = \boldsymbol{r}_k^T \boldsymbol{r}_k \tag{61}$$

Recalling the A-orthogonality of $\boldsymbol{p}_k$ and any previous $\boldsymbol{p}_j$, it is also easy to observe that $\boldsymbol{p}_k^T A\boldsymbol{p}_k = \boldsymbol{r}_k^T A\boldsymbol{p}_k$. Then, using the recurrent relation (49) to express $A\boldsymbol{p}_k$ yields:

$$\boldsymbol{p}_k^T A\boldsymbol{p}_k = \frac{1}{\alpha_k} \boldsymbol{r}_k^T \left(\boldsymbol{r}_k - \boldsymbol{r}_{k+1} \right) = \frac{1}{\alpha_k} \boldsymbol{r}_k^T \boldsymbol{r}_k \tag{62}$$

Similarly, we obtain:

$$\boldsymbol{r}_{k+1}^T A\boldsymbol{p}_k = \frac{1}{\alpha_k}\boldsymbol{r}_{k+1}^T(\boldsymbol{r}_k - \boldsymbol{r}_{k+1}) = -\frac{1}{\alpha_k}\boldsymbol{r}_{k+1}^T\boldsymbol{r}_{k+1} \tag{63}$$

Using equation (61) through (63) the scalar parameters α_k and β_k can be also computed as:

$$\alpha_k = \frac{\boldsymbol{r}_k^T\boldsymbol{r}_k}{\boldsymbol{r}_k^T A\boldsymbol{p}_k} \tag{64}$$

$$\beta_k = \frac{\boldsymbol{r}_{k+1}^T\boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T\boldsymbol{r}_k} \tag{65}$$

Introducing the new scalar parameter $\rho_k = \boldsymbol{r}_k^T\boldsymbol{r}_k$, an equivalent variant to Algorithm 2 is the following:

ALGORITHM 3: CONJUGATE GRADIENT

1. Choose $\boldsymbol{x}_0$
2. Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$ and set $\boldsymbol{p}_0 = \boldsymbol{r}_0$
3. Compute $\rho_0 = \boldsymbol{r}_0^T\boldsymbol{r}_0$
4. DO $k = 0, \ldots$ until convergence
5. $\quad \alpha_k = \rho_k / \boldsymbol{r}_k^T A\boldsymbol{p}_k$
6. $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$
7. $\quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k$
8. $\quad \rho_{k+1} = \boldsymbol{r}_{k+1}^T\boldsymbol{r}_{k+1}$
9. $\quad \beta_k = \rho_{k+1}/\rho_k$
10. $\quad \boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k\boldsymbol{p}_k$
11. END DO

The CG convergence is controlled by an inequality similar to that obtained for SD. In particular, the following result holds true:

THEOREM. *Let A be symmetric and positive definite. Then the energy A-norm of $\boldsymbol{e}_k$ generated by Algorithm 2 satisfy the inequality:*

$$\|\boldsymbol{e}_k\|_A \leq \mu_1^k \|\boldsymbol{e}_0\|_A \tag{66}$$

with $\mu_1 = \frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}$.

PROOF. The result (66) can be obtained by using the Chebyshev polynomials. For details, see Saad (2003), *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, pp. 198–205. □

Similarly to the SD scheme, the convergence of CG can be very fast only if the conditioning number of A is close to 1. In particular, the inequality (66) can be useful to provide an estimate of the number of iterations required to decrease the energy A-norm of the initial error by p orders of magnitude:

$$\frac{\|\boldsymbol{e}_k\|_A}{\|\boldsymbol{e}_0\|_A} \leq 10^{-p} \qquad \Rightarrow \qquad \left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^k = 10^{-p} \tag{67}$$

Taking the natural logarithms at both sides of the rightmost equation (67) and using the first-order Taylor expansion:

$$\ln \frac{\sqrt{\kappa\left(A\right)}-1}{\sqrt{\kappa\left(A\right)}+1} \simeq \frac{-2}{\sqrt{\kappa\left(A\right)}+1} \tag{68}$$

we obtain:

$$k \simeq \frac{p \ln 10}{2}\left(\sqrt{\kappa\left(A\right)}+1\right) \tag{69}$$

i.e., the number of iterations required to decrease the energy A-norm of the initial error by p orders of magnitude is proportional to the square root of $\kappa(A)$.

2.2.1 The Preconditioned CG (PCG)

For a long time, the actual potential of the CG method for solving symmetric positive definite linear systems went underrated. The main reason relies on the fact that in view of its finite termination property CG was initially regarded as a direct method able to provide the exact solution after at most n iterations, with n the size of matrix A. The theoretical maximum number of iterations needed to achieve $\boldsymbol{h}$ is actually equal to the number of distinct eigenvalues of A. However, as a direct method the CG algorithm is not competitive with a classical Gaussian elimination, except from a few lucky cases. In fact, the approximate computational cost of each iteration of CG is of the order of n^2 operations for the matrix-vector product and $3n$ for the three scalar products. Stopping the algorithm after n iterations produces an overall cost of about $n^3 + 3n^2$ operations which is superior to that of a standard Gaussian elimination. Moreover, it was soon clear that the finite termination property is only theoretical, as the actual number of iterations needed to achieve convergence can be also equal to $10n$ because of the rounding errors committed in finite arithmetics. Hence, the CG method can have a practical interest only if a sufficiently good accuracy can be obtained in a number of iterations much smaller than n. This result can be obtained by using an appropriate *preconditioner*.

Recalling equation (69), the error in the CG algorithm can decrease rapidly if the conditioning number of the system matrix is close to 1, i.e., the eigenvalues of A are clustered around a non-zero value. For example, such a condition occurs if the system matrix resembles the identity from a spectral point of view. The basic idea of preconditioning relies on transforming the original problem $A\boldsymbol{x} = \boldsymbol{b}$ into an equivalent one $B\boldsymbol{y} = \boldsymbol{c}$ such that the eigenspectrum of B be much more favorable for a CG iteration. Let X^{-1} be a non singular symmetric positive definite matrix. The native system (1) can be re-written as:

$$X^{-1}AX^{-1}X\boldsymbol{x} = X^{-1}\boldsymbol{b} \tag{70}$$

where we can set $B = X^{-1}AX^{-1}$, $\boldsymbol{y} = X\boldsymbol{x}$ and $\boldsymbol{c} = X^{-1}\boldsymbol{b}$. Notice that the matrix B is still symmetric positive definite, hence the CG method can be applied. The Preconditioned CG (PCG) algorithm consists of re-formulating the Algorithm 2 for the system (70) with

the new variables:

$$\boldsymbol{y}_k = X\boldsymbol{x}_k \tag{71}$$
$$\boldsymbol{p}'_k = X\boldsymbol{p}_k \tag{72}$$
$$\boldsymbol{r}'_k = \boldsymbol{c} - B\boldsymbol{y}_k = X^{-1}\boldsymbol{r}_k \tag{73}$$

The new scalar coefficient α'_k computed to minimize Φ along the search direction $\boldsymbol{p}'_k$ reads:

$$\alpha'_k = \frac{\boldsymbol{r}'^T_k \boldsymbol{p}'_k}{\boldsymbol{p}'^T_k B\boldsymbol{p}'_k} = \frac{\boldsymbol{r}^T_k \boldsymbol{p}_k}{\boldsymbol{p}^T_k A\boldsymbol{p}_k} = \alpha_k \tag{74}$$

while the recurrent relationship to update the system solution:

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \alpha'_k \boldsymbol{p}'_k \tag{75}$$

using equations (71) and (74) becomes:

$$X\boldsymbol{x}_{k+1} = X\boldsymbol{x}_k + \alpha_k X\boldsymbol{p}_k \tag{76}$$

i.e., still equation (41) if we pre-multiply both sides of (76) by X^{-1}. Similarly, the recurrent relation to update the preconditioned residual vector is:

$$\boldsymbol{r}'_{k+1} = \boldsymbol{r}'_k - \alpha'_k B\boldsymbol{p}'_k \tag{77}$$

i.e.:

$$X^{-1}\boldsymbol{r}_{k+1} = X^{-1}\boldsymbol{r}_k - \alpha_k X^{-1}AX^{-1}X\boldsymbol{p}_k \tag{78}$$

that again coincides with equation (49) if we pre-multiply both sides of (78) by X. Finally, we need the recurrent relation to update the search direction. The new scalar coefficient β'_k that prescribes the A-orthogonality between $\boldsymbol{p}'_{k+1}$ and $\boldsymbol{p}'_k$ is:

$$\beta'_k = -\frac{\boldsymbol{r}'^T_{k+1} B\boldsymbol{p}'_k}{\boldsymbol{p}'_k B\boldsymbol{p}'_k} = -\frac{\boldsymbol{r}^T_{k+1} M^{-1} A\boldsymbol{p}_k}{\boldsymbol{p}^T_k A\boldsymbol{p}_k} \tag{79}$$

where $M^{-1} = X^{-1}X^{-1}$. As a consequence, the recurrent relation for computing $\boldsymbol{p}'_{k+1}$:

$$\boldsymbol{p}'_{k+1} = \boldsymbol{r}'_{k+1} + \beta'_k \boldsymbol{p}'_k \tag{80}$$

reads:

$$X\boldsymbol{p}_{k+1} = X^{-1}\boldsymbol{r}_{k+1} + \beta'_k X\boldsymbol{p}_k \tag{81}$$

Pre-multiplying both sides of (81) by X^{-1} we obtain:

$$\boldsymbol{p}_{k+1} = M^{-1}\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k \tag{82}$$

In other words, as compared to Algorithm 2 the PCG scheme simply requires the additional computation of $M^{-1}\boldsymbol{r}_{k+1}$ at each iteration. Therefore, the PCG algorithm can be summarized as follows:

Algorithm 4: Preconditioned Conjugate Gradient

1. Choose $\boldsymbol{x}_0$ and M^{-1}
2. Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$ and set $\boldsymbol{p}_0 = M^{-1}\boldsymbol{r}_0$
3. Do $k = 0, \ldots$ until convergence
4. $\quad \alpha_k = \boldsymbol{r}_k^T \boldsymbol{p}_k / \boldsymbol{p}_k^T A \boldsymbol{p}_k$
5. $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
6. $\quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k$
7. $\quad \beta_k = -\boldsymbol{r}_{k+1}^T M^{-1} A \boldsymbol{p}_k / \boldsymbol{p}_k^T A \boldsymbol{p}_k$
8. $\quad \boldsymbol{p}_{k+1} = M^{-1}\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$
9. End Do

Quite obviously, the cost per iteration increases because of the computation of $M^{-1}\boldsymbol{r}_{k+1}$. Hence, the PCG algorithm is convenient if it produces an accurate approximation of $\boldsymbol{h}$ within a number of iterations much smaller than n. To this aim, M^{-1} must be selected so as to cluster as much as possible the eigenvalues of B away from zero, e.g., around unity. As B is similar to AM^{-1}, our objective is to select M^{-1} such that:

$$AM^{-1} \simeq I \qquad \Rightarrow \qquad M^{-1} \simeq A^{-1} \tag{83}$$

The selection of a good preconditioner is the key factor for the success of the CG algorithm which is currently the method of choice for large sparse symmetric positive definite systems. There are several different ways for accomplishing the condition (83) taking into account that the computation of both M^{-1} and $M^{-1}\boldsymbol{r}_{k+1}$ should be as cheap as possible. For a recent review of this topic, see Ferronato (2012), *Preconditioning for sparse linear systems at the dawn of the 21st century: history, current developments, and future perspectives*, ISRN Applied Mathematics, 49 pages, doi: 10.5402/2012/127647.

3 Iterative methods for general systems

The PCG method converges only on symmetric positive definite matrices. In fact, removing such a hypothesis does no longer guarantees that $\Phi(\boldsymbol{x})$ as defined in (19) has a unique absolute minimum coinciding with the solution vector $\boldsymbol{h}$. Computational experiences show that it is still possible to have convergence of PCG with either "slightly" non symmetric matrices or special preconditioners defined for indefinite saddle-point problems. For general matrices, however, PCG cannot be used. This is why several attempts for generalizing PCG also for non symmetric systems have been done in the past years.

The simplest idea for using PCG on general matrices relies on symmetrizing the system by pre-multiplying both sides of (1) by A^T:

$$A^T A \boldsymbol{x} = A^T \boldsymbol{b} \tag{84}$$

The new problem (84) is denoted as *system of normal equations*. This problem, however, is usually much more ill-conditioned than (1). It is well-known that the eigenvalues of $A^T A$ satisfy the inequality:

$$|\lambda_n(A)|^2 \leq \lambda_i\left(A^T A\right) \leq |\lambda_1(A)|^2 \qquad \forall\, i = 1, \ldots, n \tag{85}$$

hence are spread over a much larger interval than those of A. Moreover, A^TA can be much denser than A and too expensive to compute and store. This is not a problem for the PCG algorithm in itself, as we just need a rule to compute the product between the system matrix and a vector. However, the actual difficulty relies in computing a good preconditioner for a matrix whose explicit form is not known.

The most successful approach is based on approximating $\boldsymbol{h}$ in proper subspaces of $\mathbb{R}^n$ with an increasing size, selecting somewhat an "optimal" solution into each subspace. As the size of the subspaces grows the approximate solution gets closer and closer to $\boldsymbol{h}$. The subspaces where searching the approximate solution can be selected by splitting $\mathbb{R}^n$ with the aid of *projection operators*.

3.1 The Generalized Minimal Residual (GMRES)

The Generalized Minimal Residual (GMRES) method was introduced in 1986 by Saad and Schultz. It is based on finding the approximate solution $\boldsymbol{x}_m$ into a subspace $\mathcal{K}_m \subset \mathbb{R}^n$ of size $m < n$ by minimizing the 2-norm of the current residual vector $\boldsymbol{r}_m$. The subspace $\mathcal{K}_m$ where $\boldsymbol{x}_m$ is found is defined as:

$$\mathcal{K}_m = \text{ span}\left\{\boldsymbol{r}_0, A\boldsymbol{r}_0, A^2\boldsymbol{r}_0, \ldots, A^{m-1}\boldsymbol{r}_0\right\} \tag{86}$$

where, as usual, $\boldsymbol{r}_0$ is the residual vector associated to an arbitrary initial guess $\boldsymbol{x}_0$. The space defined in equation (86) is also denoted as $\mathcal{K}_m(A, \boldsymbol{r}_0)$ and is called *Krylov subspace* of size m generated by A and $\boldsymbol{r}_0$. To exploit also the possible knowledge of a "good" initial guess $\boldsymbol{x}_0$, we look for the approximation $\boldsymbol{x}_m$ belonging to $\boldsymbol{x}_0 + \mathcal{K}_m(A, \boldsymbol{r}_0)$. Therefore, the problem we aim at solving can be formally re-casted as follows. Find $\boldsymbol{x}_m$ as:

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + \boldsymbol{y}, \qquad \boldsymbol{y} \in \mathcal{K}_m\left(A, \boldsymbol{r}_0\right) \tag{87}$$

such that:

$$\|\boldsymbol{r}_m\|_2 = \|\boldsymbol{b} - A\boldsymbol{x}_m\|_2 = \min \tag{88}$$

over all vectors $\boldsymbol{x}_m$ given in the form (87). Starting from the Krylov subspace of size 1 generated by $\boldsymbol{r}_0$ and increasing m at each iteration, the solution to the problem (87) subject to the condition (88) generates a sequence of vectors $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, …, $\boldsymbol{x}_m$ converging to $\boldsymbol{h}$. Quite obviously, when $m = n$ the condition (88) produces exactly $\boldsymbol{h}$, hence similarly to CG the GMRES method is characterized by the finite termination property.

As $\boldsymbol{y}$ belongs to $\mathcal{K}_m(A, \boldsymbol{r}_0)$ it can be written as a linear combination of the basis vectors:

$$\boldsymbol{y} = c_1\boldsymbol{r}_0 + c_2A\boldsymbol{r}_0 + c_3A^2\boldsymbol{r}_0 + \ldots + c_mA^{m-1}\boldsymbol{r}_0 = \sum_{k=1}^{m} c_kA^{k-1}\boldsymbol{r}_0 \tag{89}$$

Therefore, $\boldsymbol{x}_m$ depends on the m scalar coefficients c_1, c_2, …, c_m that can be determined imposing the condition (88). Unfortunately, writing $\boldsymbol{y}$ as in (89) is not a good choice in finite arithmetics because the basis $\boldsymbol{r}_0$, $A\boldsymbol{r}_0$, …, $A^{m-1}\boldsymbol{r}_0$ is only "little linearly indepedent".

In fact, it can be proved that:

$$A^k \boldsymbol{r}_0 = \boldsymbol{u}_1 + O\left(\frac{|\lambda_2|}{|\lambda_1|}\right)^k \tag{90}$$

i.e., the vectors of the basis tend to be parallel to the eigenvector $\boldsymbol{u}_1$ associated to the first eigenvalue λ_1 of A. The result (90) can be easily proved for symmetric positive definite matrices. In this case, the eigenvectors $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, ..., $\boldsymbol{u}_n$ of A form an orthonormal basis of $\mathbb{R}^n$, so that $\boldsymbol{r}_0$ can be written as:

$$\boldsymbol{r}_0 = r_1 \boldsymbol{u}_1 + r_2 \boldsymbol{u}_2 + \ldots + r_n \boldsymbol{u}_n \tag{91}$$

Using equation (91), the general vector $A^k \boldsymbol{r}_0$ reads:

$$A^k \boldsymbol{r}_0 = r_1 \lambda_1^k \boldsymbol{u}_1 + r_2 \lambda_2^k \boldsymbol{u}_2 + \ldots + r_n \lambda_n^k \boldsymbol{u}_n \tag{92}$$

Collecting λ_1^k at the right-hand side of (92) yields:

$$A^k \boldsymbol{r}_0 = \lambda_1^k \left[r_1 \boldsymbol{u}_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^k \boldsymbol{u}_2 + \ldots + \left(\frac{\lambda_n}{\lambda_1}\right)^k \boldsymbol{u}_n \right] \tag{93}$$

that proves (90). It can be shown that the hypothesis of symmetry and positive definiteness of A can be actually removed, hence the result (90) holds true for any matrix A.

In finite arithmetics, the basis vectors for $\mathcal{K}_m(A, \boldsymbol{r}_0)$ are practically parallel to $\boldsymbol{u}_1$ already with $m \simeq 3 \div 4$. Therefore, it is desirable to express $\boldsymbol{y}$ through a more convenient basis of $\mathcal{K}_m(A, \boldsymbol{r}_0)$, e.g., using an orthonormal set of m vectors. The transformation of the native basis to an orthonormal one can be performed by using a Gram-Schmidt procedure. Assume that $\boldsymbol{g}_1$, $\boldsymbol{g}_2$, ..., $\boldsymbol{g}_m$ is a linearly independent basis for a subspace $\mathcal{G} \subset \mathbb{R}^n$ of size m. The first vector $\boldsymbol{v}_1$ of an orthormal basis of $\mathcal{G}$ is simply computed by normalizing $\boldsymbol{g}_1$:

$$\boldsymbol{v}_1 = \frac{\boldsymbol{g}_1}{\|\boldsymbol{g}_1\|_2} \tag{94}$$

Then, the second vector $\boldsymbol{v}_2$ is obtained as:

$$\hat{\boldsymbol{v}}_2 = \boldsymbol{g}_2 - \left(\boldsymbol{g}_2^T \boldsymbol{v}_1\right) \boldsymbol{v}_1, \qquad \boldsymbol{v}_2 = \frac{\hat{\boldsymbol{v}}_2}{\|\hat{\boldsymbol{v}}_2\|_2} \tag{95}$$

It is easy to prove that $\hat{\boldsymbol{v}}_2$ is orthogonal to $\boldsymbol{v}_1$. Once $\hat{\boldsymbol{v}}_2$ is available, it can be trivially normalized. Similarly, the third vector $\boldsymbol{v}_3$ is:

$$\hat{\boldsymbol{v}}_3 = \boldsymbol{g}_3 - \left(\boldsymbol{g}_3^T \boldsymbol{v}_1\right) \boldsymbol{v}_1 - \left(\boldsymbol{g}_3^T \boldsymbol{v}_2\right) \boldsymbol{v}_2, \qquad \boldsymbol{v}_3 = \frac{\hat{\boldsymbol{v}}_3}{\|\hat{\boldsymbol{v}}_3\|_2} \tag{96}$$

and so on. Generalizing equation (96) to the computation of $\boldsymbol{v}_{k+1}$ yields:

$$\hat{\boldsymbol{v}}_{k+1} = \boldsymbol{g}_{k+1} - \sum_{j=1}^{k} h_{jk} \boldsymbol{v}_j, \qquad \boldsymbol{v}_{k+1} = \frac{\hat{\boldsymbol{v}}_{k+1}}{\|\hat{\boldsymbol{v}}_{k+1}\|_2} \tag{97}$$

where each scalar coefficient h_{jk} is defined as:

$$h_{jk} = \boldsymbol{g}_{k+1}^T \boldsymbol{v}_j \tag{98}$$

For the Krylov subspace $\mathcal{K}_m(A, \boldsymbol{r}_0)$ we have:

$$\begin{aligned} \boldsymbol{g}_1 &= \boldsymbol{r}_0 && (99)\\ \boldsymbol{g}_2 &= A\boldsymbol{r}_0 = A\boldsymbol{g}_1 && (100)\\ \boldsymbol{g}_3 &= A^2\boldsymbol{r}_0 = A\boldsymbol{g}_2 && (101)\\ &\vdots \end{aligned}$$

i.e., each vector $\boldsymbol{g}_k$ can be simply obtained by a matrix-vector product with the last vector of the basis. Thus, at the $(k+1)$-th step of the Gram-Schmidt procedure $\boldsymbol{g}_{k+1}$ can be computed as $A\boldsymbol{v}_k$ and equation (97) reads:

$$\hat{\boldsymbol{v}}_{k+1} = A\boldsymbol{v}_k - \sum_{j=1}^{k} h_{jk}\boldsymbol{v}_j, \qquad \boldsymbol{v}_{k+1} = \frac{\hat{\boldsymbol{v}}_{k+1}}{\|\hat{\boldsymbol{v}}_{k+1}\|_2} \tag{102}$$

with:

$$h_{jk} = \boldsymbol{v}_k^T A^T \boldsymbol{v}_j = \boldsymbol{v}_j^T A \boldsymbol{v}_k \tag{103}$$

Recalling that $\hat{\boldsymbol{v}}_{k+1} = \|\hat{\boldsymbol{v}}_{k+1}\|_2 \boldsymbol{v}_{k+1}$, equation (102) can be also written as:

$$\|\hat{\boldsymbol{v}}_{k+1}\|_2 \, \boldsymbol{v}_{k+1} = A\boldsymbol{v}_k - \sum_{j=1}^{k} h_{jk}\boldsymbol{v}_j \tag{104}$$

Pre-multiplying both sides of equation (104) by $\boldsymbol{v}_{k+1}^T$ and exploiting the fact that all vectors $\boldsymbol{v}_j$ are orthonormal by construction, we achieve the following result:

$$\|\hat{\boldsymbol{v}}_{k+1}\|_2 = \boldsymbol{v}_{k+1}^T A \boldsymbol{v}_k = h_{k+1,k} \tag{105}$$

On summary, the Gram-Schmidt procedure to build an orthonormal basis for $\mathcal{K}_m(A, \boldsymbol{r}_0)$ proceeds as follows:

ALGORITHM 5: GRAM-SCHMIDT

1. Set $\beta = \|\boldsymbol{r}_0\|_2$ and $\boldsymbol{v}_1 = \boldsymbol{r}_0/\beta$
2. DO $k = 1, \ldots, m$
3. DO $j = 1, \ldots, k$
4. $h_{jk} = \boldsymbol{v}_j^T A \boldsymbol{v}_k$
5. END DO
6. $\hat{\boldsymbol{v}}_{k+1} = A\boldsymbol{v}_k - \sum_{j=1}^{k} h_{jk}\boldsymbol{v}_j$
7. $h_{k+1,k} = \|\hat{\boldsymbol{v}}_{k+1}\|_2$
8. $\boldsymbol{v}_{k+1} = \hat{\boldsymbol{v}}_{k+1}/h_{k+1,k}$
9. END DO

It is well-known that the standard Gram-Schmidt procedure is computationally unstable as it can soon produce non-orthogonal vectors because of rounding errors. There are several different alternative formulations which are mathematically equivalent to Algorithm 5 but possess better numerical properties. One of the most popular Modified Gram-Schmidt procedure is the following:

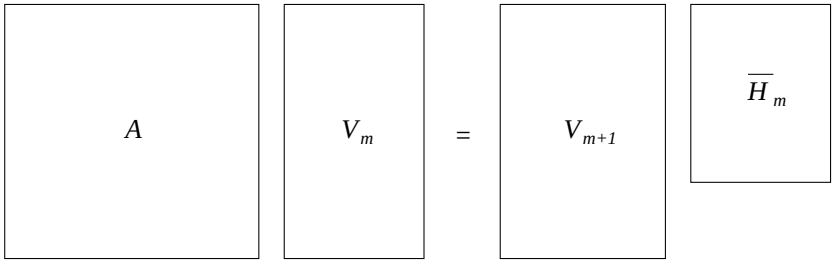

Figure 5: Schematic representation of equation (113).

ALGORITHM 6: MODIFIED GRAM-SCHMIDT

1. Set $\beta = \|\boldsymbol{r}_0\|_2$ and $\boldsymbol{v}_1 = \boldsymbol{r}_0/\beta$
2. DO $k = 1, \ldots, m$
3. $\quad \boldsymbol{w}_{k+1} = A\boldsymbol{v}_k$
4. $\quad$ DO $j = 1, \ldots, k$
5. $\quad\quad h_{jk} = \boldsymbol{w}_{k+1}^T \boldsymbol{v}_j$
6. $\quad\quad \boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_{k+1} - h_{jk}\boldsymbol{v}_j$
7. $\quad$ END DO
8. $\quad h_{k+1,k} = \|\boldsymbol{w}_{k+1}\|_2$
9. $\quad \boldsymbol{v}_{k+1} = \boldsymbol{w}_{k+1}/h_{k+1,k}$
10. END DO

Using either Algorithm 5 or 6 we are able to compute an orthonormal basis for $\mathcal{K}_m(A, \boldsymbol{r}_0)$. The vector $\boldsymbol{y}$ of equation (87) can now be written as:

$$\boldsymbol{y} = z_1\boldsymbol{v}_1 + z_2\boldsymbol{v}_2 + \ldots + z_m\boldsymbol{v}_m = V_m\boldsymbol{z} \tag{106}$$

where $\boldsymbol{z} \in \mathbb{R}^m$ and V_m is the $n \times m$ matrix with the vectors $\boldsymbol{v}_j$, $j = 1, m$, as columns. Hence, the problem to be solved at each GMRES iteration can be re-formulated as follows. Find $\boldsymbol{z} \in \mathbb{R}^m$ such that $\|\boldsymbol{r}_m\|_2$ is minimum over the current Krylov subspace $\mathcal{K}_m(A, \boldsymbol{r}_0)$.

Let us consider again equation (102) where $\hat{\boldsymbol{v}}_{k+1}$ is written using (105):

$$h_{k+1,k}\boldsymbol{v}_{k+1} = A\boldsymbol{v}_k - \sum_{j=1}^{k} h_{jk}\boldsymbol{v}_j \tag{107}$$

Extending the summation at the right-hand side to the left-hand side we have:

$$A\boldsymbol{v}_k = \sum_{j=1}^{k+1} h_{jk}\boldsymbol{v}_j \tag{108}$$

that for $k = 1, \ldots, m$ explicitly reads:

$$
\begin{aligned}
A\boldsymbol{v}_1 &= h_{11}\boldsymbol{v}_1 + h_{21}\boldsymbol{v}_2 && (109)\\
A\boldsymbol{v}_2 &= h_{12}\boldsymbol{v}_1 + h_{22}\boldsymbol{v}_2 + h_{32}\boldsymbol{v}_3 && (110)\\
A\boldsymbol{v}_3 &= h_{13}\boldsymbol{v}_1 + h_{23}\boldsymbol{v}_2 + h_{33}\boldsymbol{v}_3 + h_{43}\boldsymbol{v}_4 && (111)\\
&\vdots \\
A\boldsymbol{v}_m &= h_{1m}\boldsymbol{v}_1 + h_{2m}\boldsymbol{v}_2 + \ldots + h_{mm}\boldsymbol{v}_m + h_{m+1,m}\boldsymbol{v}_{m+1} && (112)
\end{aligned}
$$

Using the matrix V_m as defined above, equations (109)-(112) can be also written in the compact matrix form as (Figure 5):

$$AV_m = V_{m+1}\overline{H}_m \tag{113}$$

where $\overline{H}_m$ is an $(m+1) \times m$ matrix with an upper Hessemberg form collecting all the scalars h_{jk}:

$$\overline{H}_m = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \cdots & h_{1m} \\ h_{21} & h_{22} & h_{23} & & h_{2m} \\ 0 & h_{32} & h_{33} & & h_{3m} \\ 0 & 0 & h_{43} & & h_{4m} \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & h_{m+1,m} \end{bmatrix} \tag{114}$$

Using equations (87) and (106) yields the following expression for the residual vector $\boldsymbol{r}_m$:

$$\boldsymbol{r}_m = \boldsymbol{b} - A\left(\boldsymbol{x}_0 + V_m\boldsymbol{z}\right) = \boldsymbol{r}_0 - AV_m\boldsymbol{z} \tag{115}$$

Then, with equation (113) we have:

$$\boldsymbol{r}_m = \boldsymbol{r}_0 - V_{m+1}\overline{H}_m\boldsymbol{z} \tag{116}$$

Recall also that the first column of V_{m+1} is $\boldsymbol{v}_1$, i.e., $\boldsymbol{r}_0/\beta$, where $\beta = \|\boldsymbol{r}_0\|_2$. Hence, the initial residual $\boldsymbol{r}_0$ can be also written as:

$$\boldsymbol{r}_0 = \beta V_{m+1}\boldsymbol{i}_1 \tag{117}$$

where $\boldsymbol{i}_1 = [1, 0, 0, \ldots, 0]^T \in \mathbb{R}^{m+1}$. Using equation (117), the residual vector at the m-th step reads:

$$\boldsymbol{r}_m = V_{m+1}\left(\beta\boldsymbol{i}_1 - \overline{H}_m\boldsymbol{z}\right) \tag{118}$$

As $V_{m+1}^T V_{m+1} = I$ by construction, the 2-norm of $\boldsymbol{r}_m$ is given by:

$$\|\boldsymbol{r}_m\|_2 = \left\|\beta\boldsymbol{i}_1 - \overline{H}_m\boldsymbol{z}\right\|_2 \tag{119}$$

The Hessemberg matrix $\overline{H}_m$ is rectangular, so generally $\overline{H}_m\boldsymbol{z} = \beta\boldsymbol{i}_1$ has no solution and $\|\boldsymbol{r}_m\|_2 \neq 0$. However, $\boldsymbol{z}$ can be computed so as to minimize of the right-hand side of equation (119), i.e., by solving a least-square problem of size m. This task is very cheap

if m is much smaller than n, e.g., $m \leq 100$. The minimization of $\|\boldsymbol{r}_m\|_2$ is performed by deriving the right-hand side of (119) with respect to $\boldsymbol{z}$ and setting to 0:

$$\frac{\partial}{\partial \boldsymbol{z}} \left\| \beta \boldsymbol{i}_1 - \overline{H}_m \boldsymbol{z} \right\|_2^2 = 2\overline{H}_m^T \left(\beta \boldsymbol{i}_1 - \overline{H}_m \boldsymbol{z} \right) = 0 \tag{120}$$

i.e., solving an $m \times m$ system of normal equations:

$$\overline{H}_m^T \overline{H}_m \boldsymbol{z} = \beta \overline{H}_m^T \boldsymbol{i}_1 \tag{121}$$

As is well-known, any system of normal equations is ill-conditioned. An efficient way for copying with this problem relies on using a QR factorization of $\overline{H}_m$, obtained either with a Modified Gram-Schmidt or a Householder process. In both cases, $\overline{H}_m$ is decomposed as:

$$\overline{H}_m = QR \tag{122}$$

where Q is an $(m+1) \times (m+1)$ orthogonal matrix and R is an $(m+1) \times m$ "quasi-upper triangular" matrix in the form:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & \cdots & r_{1m} \\ 0 & r_{22} & r_{23} & & r_{2m} \\ 0 & 0 & r_{33} & & r_{3m} \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & r_{mm} \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} = \begin{bmatrix} \tilde{R} \\ 0 \end{bmatrix} \tag{123}$$

i.e., an $m \times m$ upper triangular matrix $\tilde{R}$ plus a null row. Introducing the factorization (122) into equation (121) yields:

$$R^T R \boldsymbol{z} = R^T \left(\beta Q^T \boldsymbol{i}_1 \right) \tag{124}$$

that is the solution to the least-square problem:

$$\boldsymbol{z} = \ \text{argmin}_{\boldsymbol{t}} \, \| \boldsymbol{d} - R\boldsymbol{t} \|_2^2 \tag{125}$$

where $\boldsymbol{d} = \beta Q^T \boldsymbol{i}_1$. As the last row of R is null, the minimum of $\|\boldsymbol{d} - R\boldsymbol{z}\|_2$ is simply obtained when:

$$\tilde{R} \boldsymbol{z} = \tilde{\boldsymbol{d}} \tag{126}$$

with $\tilde{\boldsymbol{d}}$ the vector collecting the first m components of $\boldsymbol{d}$. The system (126) is quite small and upper triangular, so its solution is trivial and relatively inexpensive. Finally, note that the 2-norm of the current residual vector is immediately available as the last component of $\boldsymbol{d}$. In fact, using equation (123) and recalling the condition (126) lead to:

$$\|\boldsymbol{r}_m\|_2 = \left\| \begin{bmatrix} \tilde{\boldsymbol{d}} \\ d_{m+1} \end{bmatrix} - \begin{bmatrix} \tilde{R} \\ 0 \end{bmatrix} \boldsymbol{z} \right\|_2 = |d_{m+1}| \tag{127}$$

On summary, the overall GMRES algorithm runs as follows:

Algorithm 7: Generalized Minimal Residual

1. Choose $\boldsymbol{x}_0$
2. Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
3. Set $\beta = \|\boldsymbol{r}_0\|_2$ and $\boldsymbol{v}_1 = \boldsymbol{r}_0/\beta$
4. Do $k = 1, \ldots$ until convergence
5. Update V_k
6. $\boldsymbol{w}_{k+1} = A\boldsymbol{v}_k$
7. Do $j = 1, \ldots, k$
8. $h_{jk} = \boldsymbol{w}_{k+1}^T \boldsymbol{v}_j$
9. $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_{k+1} - h_{jk}\boldsymbol{v}_j$
10. End Do
11. $h_{k+1,k} = \|\boldsymbol{w}_{k+1}\|_2$
12. $\boldsymbol{v}_{k+1} = \boldsymbol{w}_{k+1}/h_{k+1,k}$
13. Update $\overline{H}_k$
14. Compute Q and R such that $\overline{H}_k = QR$
15. $\boldsymbol{d} = \beta Q^T \boldsymbol{i}_1$
16. Solve $\tilde{R}\boldsymbol{z} = \tilde{\boldsymbol{d}}$
17. $\boldsymbol{x}_k = \boldsymbol{x}_0 + V_k\boldsymbol{z}$
18. End Do

Notice that in Algorithm 7 the computation of the approximate solution in line 17 could be actually performed only at the end of the iterative process. In fact, the current residual norm is known independently of $\boldsymbol{x}_k$ from the computation of $\boldsymbol{d}$ in line 15.

3.1.1 Computational issues

GMRES is an optimal iterative method in the sense that it satisfies a minimization property at each step of the procedure, hence the solution is somewhat the "best" one at every iteration. This implies the finite termination property which ensures that the exact solution is achieved in infinite precision after at most n iterations. The only possible breakdown in Algorithm 7 can occur if $h_{k+1,k} = 0$, i.e., the new vector computed to enlarge the current Krylov subspace is null. Such an occurrence means that the size of $\mathcal{K}_k$ cannot be further increased, hence the solution $\boldsymbol{h}$ must lie in $\mathcal{K}_k$ and $\boldsymbol{x}_k = \boldsymbol{h}$. This is called *lucky breakdown* of the GMRES algorithm, as convergence is achieved in less than n steps.

The convergence rate of GMRES cannot be easily related to the eigenvalue distribution of A as in the CG method. Assuming that A is diagonalizable:

$$A = U\Lambda U^{-1} \tag{128}$$

with $\Lambda, U \in \mathbb{C}^{n\times n}$, it can be proved by using the Chebyshev polynomials that after k GMRES iterations the ratio between the current and the initial 2-norms of the residual vector satisfy the following inequality:

$$\frac{\|\boldsymbol{r}_k\|_2}{\|\boldsymbol{r}_0\|_2} \leq \kappa(U) \min_{\mathbb{P}_k} \|p_k(\Lambda)\|_\infty \tag{129}$$

where $\mathbb{P}_k$ is the space of polynomials of degree k and $p_k \in \mathbb{P}_k$. The upper bound (129) shows that convergence of GMRES depends not only on the distribution of the eigenvalues of A, but also on the conditioning number of the matrix of the eigenvectors. Nevertheless, the computational experience shows that in engineering problems arising from the discretization of PDEs a good clustering of the eigenvalues of A away from the origin of the complex plane usually denotes a favorable condition for a fast convergence. This is why preconditioning can be effectively used also in the GMRES algorithm, though its role is not as clear as in the PCG method.

In the GMRES method two kinds of preconditioning can be used:

1. *left preconditioning*:
$$M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b} \tag{130}$$

2. *right preconditioning*:
$$AM^{-1}\boldsymbol{y} = \boldsymbol{b}, \qquad \boldsymbol{x} = M^{-1}\boldsymbol{y} \tag{131}$$

Right preconditioning is generally preferable, as with system (130) the preconditioned residual is actually minimized:
$$\left\|M^{-1}\boldsymbol{r}_k\right\|_2 = \min \tag{132}$$
with no control on the size of the true residual. If the preconditioner M^{-1} is available in a factored form, then *split preconditioning* is also possible coupling both left and right preconditioning. Applying the Algorithm 7 on either system (130) or (131) implies adding the additional vector $\boldsymbol{s} = M^{-1}\boldsymbol{w}_{k+1}$ for the computation of h_{jk}:
$$h_{jk} = \boldsymbol{v}_j^T M^{-1}\boldsymbol{w}_{k+1} = \boldsymbol{s}^T\boldsymbol{v}_j \tag{133}$$

From the computational point of view, GMRES is a very expensive method for both the required storage and the number of operations. As far as the storage is concerned, at each GMRES iteration all vectors $\boldsymbol{v}_k$ forming the basis of the current Krylov subspace must be kept in memory. For large values of k, e.g., on the order of hundreds, the required storage may become really significant. In any case, reaching a number of iterations of the order of the matrix size n, as it would be required by the finite termination property, is thouroughly unfeasible. As to the computational cost, the number of operations required by the Modified Gram-Schmidt increases with the iteration count. Moreover, the size of $\overline{H}_k$ and $\tilde{R}$ grows at each iteration, and the related cost may become a price too high to pay. This is the consequence of the fact that GMRES is a *long-term recurrence*, i.e., it needs an increasing number of vectors to compute the sequence of approximate solutions.

To reduce the computational cost of GMRES some practical implementations have been advanced. The most popular solution relies on setting a maximum number p of vectors $\boldsymbol{v}_k$ to store. At the $(p+1)$-th iteration a new Krylov subspace is built starting from the last residual $\boldsymbol{r}_p$. This variant gives rise to the Restarted GMRES, usually denoted as GMRES(p). The advantage of GMRES(p) is that the storage and the computational cost

of each iteration remain limited. However, the optimal GMRES properties are theoretically lost, so that GMRES(p) is no longer guaranteed to converge.

3.2 Projection methods

The basic idea of projection techniques consists of extracting an approximate solution to problem (1) from $\mathcal{K}_m(A, \boldsymbol{r}_0)$ by defining a projection operator that depends on m constraints. A typical way to do so is to prescribe m independent orthogonality conditions on the residual vector $\boldsymbol{r}_m$, thus implicitly defining another subspace $\mathcal{L}_m$. In other words, we seek an approximate solution in $\mathcal{K}_m(A, \boldsymbol{r}_0)$ such that the residual vector is orthogonal to $\mathcal{L}_m$. Such a mathematical framework is also known as *Petrov-Galerkin condition*.

As in the GMRES method, we wish to exploit the knowledge of an initial guess solution $\boldsymbol{x}_0$. The approximate solution $\boldsymbol{x}_m$ is found in the affine subspace $\boldsymbol{x}_0 + \mathcal{K}_m(A, \boldsymbol{r}_0)$ as in equation (87) where $\boldsymbol{y}$ is chosen by imposing that:

$$\boldsymbol{r}_m = \boldsymbol{b} - A\boldsymbol{x}_m \perp \mathcal{L}_m \tag{134}$$

Using (87), the orthogonality condition (134) depends explicitly on $\boldsymbol{y}$:

$$\boldsymbol{r}_0 - A\boldsymbol{y} \perp \mathcal{L}_m \tag{135}$$

Let $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$ form a basis for $\mathcal{K}_m(A, \boldsymbol{r}_0)$ and $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m$ a basis for $\mathcal{L}_m$. Writing $\boldsymbol{y}$ as in equation (106) the orthogonality condition (135) can be refomulated using the basis $\boldsymbol{w}_i$:

$$\boldsymbol{w}_i^T (\boldsymbol{r}_0 - A\boldsymbol{y}) = 0 \qquad i = 1, \ldots, m \tag{136}$$

Similarly to V_m, define W_m as the $(n \times m)$ matrix whose columns are the vectors $\boldsymbol{w}_i$. The m conditions (136) now read:

$$W_m^T (\boldsymbol{r}_0 - AV_m\boldsymbol{z}) = 0 \tag{137}$$

Equation (137) defines a linear system in the unknown vector $\boldsymbol{z}$. Assuming that the matrix $W_m^T AV_m$ is nonsingular, the approximate solution $\boldsymbol{x}_m$ is:

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m \left(W_m^T AV_m\right)^{-1} W_m^T \boldsymbol{r}_0 \tag{138}$$

The prototype for a general projection algorithm is the following:

ALGORITHM 8: PROTOTYPE PROJECTION METHOD

1. Select a pair of subspaces $\mathcal{K}_m(A, \boldsymbol{r}_0)$ and $\mathcal{L}_m$
2. DO $m = 1, \ldots$ until convergence
3. Compute $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$
4. Compute $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m$
5. $\boldsymbol{z} = \left(W_m^T AV_m\right)^{-1} W_m^T \boldsymbol{r}_0$
6. $\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m \boldsymbol{z}$
7. END DO

Algorithm 8 requires the selection of the space $\mathcal{L}_m$. The most successful approach is based on the use of the Krylov subspace associated with the transposed of A:

$$\mathcal{L}_m\left(A^T, \boldsymbol{r}_0\right) = \text{ span}\left\{\boldsymbol{r}_0, A^T\boldsymbol{r}_0, \left(A^T\right)^2\boldsymbol{r}_0, \ldots, \left(A^T\right)^{m-1}\boldsymbol{r}_0\right\} \tag{139}$$

A projection method where $\mathcal{K}_m = \mathcal{L}_m$ is called orthogonal. The iterative techniques developed using $\mathcal{L}_m$ as defined in (139) are therefore called *nonorthogonal projection methods onto Krylov subspaces*.

Nonorthogonal projection methods have a number of appealing properties and are much used in practice, however they are quite hard to analyze theoretically. In particular, it is not possible to prove that the current solution $\boldsymbol{x}_m$ is optimal in some sense, so these methods are intrinsically less robust than CG and GMRES. Nevertheless, the experience shows that when these methods converge they are generally faster than GMRES.

3.3 The Lanczos biorthogonalization algorithm

To make Algorithm 8 of practical interest, the linear system with matrix $W_m^T A V_m$ should be as easy to solve as possible. For example, it could be convenient to choose V_m and W_m in such a way that the matrix $W_m^T A V_m$ is tridiagonal, so that vector $\boldsymbol{z}$ can be conveniently computed with the Thomas algorithm.

Let us compute the bases $\boldsymbol{v}_i$ and $\boldsymbol{w}_i$ for $\mathcal{K}_m(A, \boldsymbol{r}_0)$ and $\mathcal{L}_m(A^T, \boldsymbol{r}_0)$, respectively, so that:

$$\boldsymbol{w}_i^T \boldsymbol{v}_j = \delta_{ij} \tag{140}$$

where δ_{ij} is the Kronecker delta. Two sequences of vectors satisfying the condition (140) are said to be *biorthogonal*. Recalling equation (102), the bases for the Krylov subspace associated to A and A^T can be calculated with the following recurrences:

$$\hat{\boldsymbol{v}}_{k+1} = A\boldsymbol{v}_k - \sum_{j=1}^{k} h_{jk}\boldsymbol{v}_j \tag{141}$$

$$\hat{\boldsymbol{w}}_{k+1} = A^T\boldsymbol{w}_k - \sum_{j=1}^{k} \tilde{h}_{jk}\boldsymbol{w}_j \tag{142}$$

The coefficients h_{jk} and $\tilde{h}_{jk}$ are obtained by imposing the newly generated vectors $\hat{\boldsymbol{v}}_{k+1}$ and $\hat{\boldsymbol{w}}_{k+1}$ to be orthogonal to $\boldsymbol{w}_j$ and $\boldsymbol{v}_j$, respectively, with $j \leq k$. This condition leads to:

$$\boldsymbol{w}_j^T \hat{\boldsymbol{v}}_{k+1} = \boldsymbol{w}_j^T A\boldsymbol{v}_k - h_{jk}\boldsymbol{w}_j^T\boldsymbol{v}_j = 0 \quad \Rightarrow \quad h_{jk} = \boldsymbol{w}_j^T A\boldsymbol{v}_k \tag{143}$$

$$\boldsymbol{v}_j^T \hat{\boldsymbol{w}}_{k+1} = \boldsymbol{v}_j^T A^T\boldsymbol{w}_k - \tilde{h}_{jk}\boldsymbol{v}_j^T\boldsymbol{w}_j = 0 \quad \Rightarrow \quad \tilde{h}_{jk} = \boldsymbol{v}_j^T A^T\boldsymbol{w}_k \tag{144}$$

Recall that all vectors at steps $j \leq k$ satisfy the hypothesis (140) by construction. As the coefficients h_{jk} and $\tilde{h}_{jk}$ have the same role as those in GMRES, they can be used to form

the matrices H and $\tilde{H}$ both possessing a Hessemberg structure. Moreover, a quick look to equations (143) and (144) shows that:

$$h_{jk} = \tilde{h}_{kj} \qquad \Rightarrow \qquad H = \tilde{H}^T \tag{145}$$

However, a Hessemberg matrix can equate the transposed of another Hessemberg matrix only if it is tridiagonal. Therefore $h_{jk} = \tilde{h}_{kj} = 0$ for all indices $j < k-1$ and equations (141) and (142) simplify into efficient three-term recurrences. Setting:

$$\alpha_k = h_{k,k} \tag{146}$$

$$\beta_k = h_{k-1,k} \tag{147}$$

$$\delta_k = \tilde{h}_{k-1,k} \tag{148}$$

equations (141) and (142) become:

$$\hat{\boldsymbol{v}}_{k+1} = A\boldsymbol{v}_k - \alpha_k \boldsymbol{v}_k - \beta_k \boldsymbol{v}_{k-1} \tag{149}$$

$$\hat{\boldsymbol{w}}_{k+1} = A^T \boldsymbol{w}_k - \alpha_k \boldsymbol{w}_k - \delta_k \boldsymbol{w}_{k-1} \tag{150}$$

The newly generated vectors $\hat{\boldsymbol{v}}_{k+1}$ and $\hat{\boldsymbol{w}}_{k+1}$ have to be scaled so that their scalar product is 1. In other terms, we seek two scalars γ_v and γ_w:

$$\boldsymbol{v}_{k+1} = \hat{\boldsymbol{v}}_{k+1}/\gamma_v \tag{151}$$

$$\boldsymbol{w}_{k+1} = \hat{\boldsymbol{w}}_{k+1}/\gamma_w \tag{152}$$

such that:

$$\boldsymbol{w}_{k+1}^T \boldsymbol{v}_{k+1} = \frac{\hat{\boldsymbol{w}}_{k+1}^T \hat{\boldsymbol{v}}_{k+1}}{\gamma_w \gamma_v} = 1 \qquad \Rightarrow \qquad \gamma_w \gamma_v = \hat{\boldsymbol{w}}_{k+1}^T \hat{\boldsymbol{v}}_{k+1} \tag{153}$$

Let us find the relationship existing between the scaling factors γ_v and γ_w and the coefficients of the three-term recurrences (149) and (150). Computing β_{k+1} and δ_{k+1} we get:

$$\beta_{k+1} = \boldsymbol{v}_{k+1}^T A^T \boldsymbol{w}_k = \boldsymbol{v}_{k+1}^T \left(\gamma_w \boldsymbol{w}_{k+1} + \alpha_k \boldsymbol{w}_k + \delta_k \boldsymbol{w}_{k-1}\right) = \gamma_w \tag{154}$$

$$\delta_{k+1} = \boldsymbol{w}_{k+1}^T A \boldsymbol{v}_k = \boldsymbol{w}_{k+1}^T \left(\gamma_v \boldsymbol{v}_{k+1} + \alpha_k \boldsymbol{v}_k + \beta_k \boldsymbol{v}_{k-1}\right) = \gamma_v \tag{155}$$

so that it is possible to update β_{k+1} and δ_{k+1} immediately after the recurrences (149) and (150). Putting all these relations together we obtain an algorithm for building a pair of bi-orthogonal bases for $\mathcal{K}_m(A, \boldsymbol{r}_0)$ and $\mathcal{L}_m(A^T, \boldsymbol{r}_0)$. Such an algorithm is known as *Lanczos biorthogonalization*:

ALGORITHM 9: LANCZOS BIORTHOGONALIZATION

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. DO $k = 1, \ldots, m$
4. $\quad \alpha_k = \boldsymbol{w}_k^T A \boldsymbol{v}_k$
5. $\quad \hat{\boldsymbol{v}}_{k+1} = A\boldsymbol{v}_k - \alpha_k \boldsymbol{v}_k - \beta_k \boldsymbol{v}_{k-1}$
6. $\quad \hat{\boldsymbol{w}}_{k+1} = A^T \boldsymbol{w}_k - \alpha_k \boldsymbol{w}_k - \delta_k \boldsymbol{w}_{k-1}$
7. $\quad$ Choose β_{k+1} and δ_{k+1} such that $\beta_{k+1}\delta_{k+1} = \hat{\boldsymbol{w}}_{k+1}^T \hat{\boldsymbol{v}}_{k+1}$
8. $\quad \boldsymbol{v}_{k+1} = \hat{\boldsymbol{v}}_{k+1}/\delta_{k+1}$
9. $\quad \boldsymbol{w}_{k+1} = \hat{\boldsymbol{w}}_{k+1}/\beta_{k+1}$
10. END DO

Note that theoretically there are infinite ways of defining β_{k+1} and δ_{k+1} to accomplish the condition in step 7 of Algorithm 9. The most usual choice is the following:

$$\delta_{k+1} = |\hat{\boldsymbol{w}}_{k+1}^T \hat{\boldsymbol{v}}_{k+1}|^{1/2} \tag{156}$$

$$\beta_{k+1} = \hat{\boldsymbol{w}}_{k+1}^T \hat{\boldsymbol{v}}_{k+1}/\delta_{k+1} \tag{157}$$

As a consequence of equations (156) and (157), the coefficients δ_k are always positive with $\beta_k = \pm\delta_k$.

The main appeal of the Lanczos biorthogonalization algorithm stems from the fact that it allows for expanding the size of $\mathcal{K}_m(A, \boldsymbol{r}_0)$ and $\mathcal{L}_m(A^T, \boldsymbol{r}_0)$ using only the last two vectors of the basis. Therefore it is not necessary to store all the vectors $\boldsymbol{v}_i$ and $\boldsymbol{w}_i$ with a huge memory saving as m increases. This is known as *short-term recurrence*. Moreover, only the coefficients α_k, β_k and δ_k are required, instead of all h_{jk}, i.e., three vectors in place of a Hessemberg matrix. By contrast, there are more chances for the algorithm to breakdown. Algorithm 9 will abort whenever the scalar product $\hat{\boldsymbol{w}}_{k+1}^T \hat{\boldsymbol{v}}_{k+1}$ is null, i.e., either $\hat{\boldsymbol{v}}_{k+1}$ or $\hat{\boldsymbol{w}}_{k+1}$ is the zero vector, or their inner product is zero even though $\hat{\boldsymbol{v}}_{k+1}$ and $\hat{\boldsymbol{w}}_{k+1}$ are both nonzero. The case $\hat{\boldsymbol{v}}_{k+1} = 0$ is a *lucky breakdown*, as in GMRES, because it means that the subspace $\mathcal{K}_m$ is invariant with A and the exact solution belongs to $\mathcal{K}_m$. The case $\hat{\boldsymbol{w}}_{k+1} = 0$ means that the subspace $\mathcal{L}_m$ is invariant with A^T. If a dual system $A^T\boldsymbol{x}^* = \boldsymbol{b}^*$ is to be solved, this implies that we have reached the exact solution $\boldsymbol{x}^*$, but unfortunately nothing can be said for the linear system (1). By contrast, the case $\hat{\boldsymbol{w}}_{k+1}^T \hat{\boldsymbol{v}}_{k+1} = 0$ with both $\hat{\boldsymbol{v}}_{k+1}$ and $\hat{\boldsymbol{w}}_{k+1}$ nonzero is a *serious breakdown* and nothing can be said again for both the original and the dual system.

The occurrence of a serious breakdown can be addressed using special *Look-ahead Lanczos algorithms* which allow for computing the pair $\hat{\boldsymbol{v}}_{k+2}$ and $\hat{\boldsymbol{w}}_{k+2}$ even if $\boldsymbol{v}_{k+1}$ and $\boldsymbol{w}_{k+1}$ are not defined. Unfortunately, the use of look-ahead strategies causes the loss of the short-term recurrences in Algorithm 9 with a consequent larger storage requirement.

In finite precision, near breakdowns occurring when the inner product $\hat{\boldsymbol{w}}_{k+1}^T \hat{\boldsymbol{v}}_{k+1}$ is small are potentially as dangerous as real breakdowns. It must be noted that, however, such occurrences are generally quite rare and their effect is in most cases not dramatic.

3.4 Nonsymmetric Lanczos algorithms

Consider the recurrence (149) with equations (151) and (155) written for $k = 1, \ldots, m$:

$$A\boldsymbol{v}_k = \beta_k \boldsymbol{v}_{k-1} + \alpha_k \boldsymbol{v}_k + \delta_{k+1} \boldsymbol{v}_{k+1} \qquad k = 1, \ldots, m \tag{158}$$

Recalling the definition of V_m and the reasoning that has led to equation (113), it can be easily recognized that equations (158) are equivalent to the matrix relationship:

$$AV_m = V_{m+1}\overline{T}_m \tag{159}$$

where $\overline{T}_m$ is the $(m+1) \times m$ "quasi-tridiagonal" matrix:

$$\overline{T}_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & & \cdots & & \\ & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \delta_m & \alpha_m \\ & & & & \delta_{m+1} \end{bmatrix} \tag{160}$$

Let us define T_m as the $m \times m$ matrix obtained neglecting the last row of $\overline{T}_m$. The product on the right-hand side of (159) can be formally written also as:

$$V_{m+1}\overline{T}_m = [V_m, \boldsymbol{v}_{m+1}] \begin{bmatrix} T_m \\ \delta_{m+1}\boldsymbol{i}_m^T \end{bmatrix} = V_m T_m + \delta_{m+1}\boldsymbol{v}_{m+1}\boldsymbol{i}_m^T \tag{161}$$

with $\boldsymbol{i}_m = [0, 0, \ldots, 0, 1]^T$ the canonical m-th reference versor of $\mathbb{R}^m$. Premultiplying equation (159) by W_m^T, with the right-hand side given by equation (161), finally yields:

$$W_m^T A V_m = T_m \tag{162}$$

where the biorthogonality of the vectors $\boldsymbol{v}_i$ and $\boldsymbol{w}_i$ has been exploited. Equation (162) proves that the use of the Lanczos biorthogonalization algorithm to build the bases of $\mathcal{K}_m$ and $\mathcal{L}_m$ in a projection method leads to the solution of a tridiagonal system of incresing size at each step of the algorithm. The matrix T_m can be also interpreted as the projection of A onto $\mathcal{K}_m$ and orthogonal to $\mathcal{L}_m$. With the above relationships, the prototype Algorithm 8 now reads:

ALGORITHM 10: TWO-SIDED LANCZOS

1. Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
2. $\beta = \|\boldsymbol{r}_0\|_2$
3. Set $\boldsymbol{v}_1 = \boldsymbol{r}_0/\beta$ and $\boldsymbol{w}_1 = \boldsymbol{v}_1$
4. DO $m = 1, \ldots$ until convergence
5. Compute $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$ and $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m$ with Algorithm 2
6. $\boldsymbol{z} = \beta T_m^{-1}\boldsymbol{i}_1$
7. $\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m\boldsymbol{z}$
8. END DO

Notice that the bases $\boldsymbol{v}_i$ and $\boldsymbol{w}_i$ play a dual role in the sense that T_m^T could be interpreted as the projection of A^T onto $\mathcal{L}_m$ and orthogonal to $\mathcal{K}_m$. Since the main cost of Algorithm 10 relies on the construction of the vectors $\boldsymbol{v}_i$ and $\boldsymbol{w}_i$, with almost the same computational effort we could obtain the solution of the dual system $A^T\boldsymbol{x}^* = \boldsymbol{b}^*$. It goes without saying that, in the frequent case where the dual system does not exist, the operations with A^T are essentially wasted.

3.4.1 Bi-Conjugate Gradient (Bi-CG)

The two-sided Lanczos algorithm can be made computationally more efficient fully exploiting the short-term recurrence of Algorithm 9 which allows for the computation of the new

vectors $\boldsymbol{v}_{k+1}$ and $\boldsymbol{w}_{k+1}$ using only two previous vectors of the sequences. Moreover, if the tridiagonal system in step 6 is solved by the Thomas algorithm a new simple recurrence can be constructed also for $\boldsymbol{y}$.

Before developing the new algorithm, let us consider the recurrent relationships of the CG method as described in Algorithm 3. Start with $k = 0$, obtaining:

$$\begin{aligned} \boldsymbol{x}_1 &= \boldsymbol{x}_0 + \alpha_0 \boldsymbol{r}_0 && (163)\\ \boldsymbol{r}_1 &= \boldsymbol{r}_0 - \alpha_0 A\boldsymbol{r}_0 && (164)\\ \boldsymbol{p}_1 &= (1+\beta_0)\,\boldsymbol{r}_0 - \alpha_0 A\boldsymbol{r}_0 && (165) \end{aligned}$$

Now use the relationships (163)-(165) to compute the new approximations for $k = 1$:

$$\begin{aligned} \boldsymbol{x}_2 &= \boldsymbol{x}_0 + [\alpha_0 + \alpha_1 (1+\beta_0)]\,\boldsymbol{r}_0 - \alpha_1\alpha_0 A\boldsymbol{r}_0 && (166)\\ \boldsymbol{r}_2 &= \boldsymbol{r}_0 - [\alpha_0 + \alpha_1 (1+\beta_0)]\,A\boldsymbol{r}_0 + \alpha_1\alpha_0 A^2\boldsymbol{r}_0 && (167)\\ \boldsymbol{p}_2 &= [1 + \beta_1 (1+\beta_0)]\,\boldsymbol{r}_0 - [\alpha_0 + \alpha_1 (1+\beta_0) + \alpha_0\beta_1]\,A\boldsymbol{r}_0 + \alpha_1\alpha_0 A^2\boldsymbol{r}_0 && (168) \end{aligned}$$

Extending the equations above it can be easily inferred that at step m the current approximate solution and the residual vector will have the form:

$$\begin{aligned} \boldsymbol{x}_m &= \boldsymbol{x}_0 + \sum_{i=1}^{m} c_i A^{i-1}\boldsymbol{r}_0 && (169)\\ \boldsymbol{r}_m &= \sum_{i=1}^{m+1} d_i A^{i-1}\boldsymbol{r}_0 && (170) \end{aligned}$$

with c_i and d_i appropriate scalar coefficients. Equation (169) shows that the current solution $\boldsymbol{x}_m$ is actually obtained summing the initial guess $\boldsymbol{x}_0$ to a vector $\boldsymbol{y}$ that belongs to $\mathcal{K}_m(A, \boldsymbol{r}_0)$. Recalling that in the CG algorithm $\boldsymbol{r}_m$ is orthogonal to all the previous residual vectors, equation (170) also suggests that actually:

$$\boldsymbol{r}_m \perp \mathcal{K}_m\,(A, \boldsymbol{r}_0) \tag{171}$$

because $\boldsymbol{r}_j \in \mathcal{K}_{j+1}(A, \boldsymbol{r}_0)$ for any $j \leq m-1$. In other words, CG can be viewed as an orthogonal projection method where $\mathcal{K}_m = \mathcal{L}_m$. Defining, as usual, a basis $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$ for $\mathcal{K}_m(A, \boldsymbol{r}_0)$, the current approximation $\boldsymbol{x}_m$ can be formally written as in equation (138) with $V_m = W_m$:

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m \left(V_m^T A V_m\right)^{-1} V_m^T \boldsymbol{r}_0 \tag{172}$$

The projection matrix $V_m^T A V_m$ is in the Hessemberg form and is symmetric, hence it is tridiagonal. Thus, it can be concluded that CG is nothing but the two-sided Lanczos algorithm for the special case of $A = A^T$. The classical recurrent relations of CG in Algorithm 3 can be reconstructed from Algorithm 10 by solving the inner tridiagonal system with the Thomas algorithm. Denote by $\tilde{T}_m$ the matrix $V_m^T A V_m$. As A is positive definite, $\tilde{T}_m$ is too, hence a real Cholesky factorization exists:

$$\tilde{T}_m^{-1} = \tilde{L}_m^{-T} \tilde{L}_m^{-1} \tag{173}$$

with:

$$\tilde{L}_m = \begin{bmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & & \\ & & \cdots & & \\ & & l_{m-1,m-2} & l_{m-1,m-1} & \\ & & & l_{m,m-1} & l_{m,m} \end{bmatrix} \tag{174}$$

Equation (172) can be written as:

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m \tilde{L}_m^{-T} \tilde{L}_m^{-1} \beta \boldsymbol{e}_1 = \boldsymbol{x}_0 + P_m \boldsymbol{a}_m \tag{175}$$

with $P_m = V_m \tilde{L}_m^{-T}$ and $\boldsymbol{a}_m = \tilde{L}_m^{-1} \beta \boldsymbol{e}_1$. Using the definition (174) of $\tilde{L}_m$ it is easy to verify that the m-th column $\boldsymbol{p}_m$ of P_m can be computed as:

$$\boldsymbol{p}_m = \frac{1}{l_{m,m}} \left(\boldsymbol{v}_m - l_{m,m-1} \boldsymbol{p}_{m-1} \right) \tag{176}$$

i.e., by a recurrent relation involving only the previous column $\boldsymbol{p}_{m-1}$ and the current basis vector $\boldsymbol{v}_m$. Using the recurrence (176) we can write equation (175) as:

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + [P_{m-1}, \boldsymbol{p}_m] \begin{bmatrix} \boldsymbol{a}_{m-1} \\ \alpha_m \end{bmatrix} = \boldsymbol{x}_{m-1} + \alpha_m \boldsymbol{p}_m \tag{177}$$

where we can recognize again the classical short-term recurrence of CG. It is also possible to verify that the vectors $\boldsymbol{p}_i$ have conjugate directions with respect to A. In fact:

$$P_m^T A P_m = \tilde{L}_m^{-1} V_m^T A V_m \tilde{L}_m^{-T} = \tilde{L}_m^{-1} \tilde{T}_m \tilde{L}_m^{-T} = I \tag{178}$$

Finally notice that a comparison of the recurrence (176) for the vectors $\boldsymbol{p}_i$ with the corresponding relationship in Algorithm 3 shows that the residual vectors $\boldsymbol{r}_i$ are parallel to $\boldsymbol{v}_i$.

As shown above for the CG algorithm, using the Thomas algorithm to solve the tridiagonal system in Algorithm 10 yields a short-term recurrence. When A is nonsymmetric the factorization (173) of T_m becomes:

$$T_m^{-1} = U_m^{-1} L_m^{-1} \tag{179}$$

and the new vectors $\boldsymbol{p}_i$ are the columns of the matrix $P_m = V_m U_m^{-1}$. A simple recurrent relation similar to (176) can be again inferred. The residual vectors $\boldsymbol{r}_i$ are still parallel to $\boldsymbol{v}_i$, but now they are orthogonalized with respect to the sequence of $\boldsymbol{w}_i$. Because of the dual role played by the bases $\boldsymbol{v}_i$ and $\boldsymbol{w}_i$, we can define the residual $\boldsymbol{r}^*$ of the dual system $A^T \boldsymbol{x}^* = \boldsymbol{b}^*$ setting $\boldsymbol{r}_i^*$ to be parallel to $\boldsymbol{w}_i$. A new set of $\boldsymbol{p}_i^*$ vectors related to the dual system are consequently obtained as the column of the matrix $P_m^* = W_m L_m^{-T}$. It is easy to prove that the vectors $\boldsymbol{p}_i$ and $\boldsymbol{p}_i^*$ are mutually A-orthogonal:

$$(P_m^*)^T A P_m = L_m^{-1} W_m^T A V_m U_m^{-1} = L_m^{-1} T_m U_m^{-1} = I \tag{180}$$

The two-sided Lanczos algorithm with the decomposition (179) for the tridiagonal inner system is therefore called *Bi-Conjugate Gradient* (Bi-CG) and was introduced by Fletcher in 1976. It can be simply written as the CG adding the recurrent relationships for the update of the dual vectors $\boldsymbol{r}_i^*$ and $\boldsymbol{p}_i^*$:

Algorithm 11: Bi-Conjugate Gradient

1. Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$ and set $\boldsymbol{r}_0^* = \boldsymbol{r}_0$
2. Set $\boldsymbol{p}_0 = \boldsymbol{r}_0$ and $\boldsymbol{p}_0^* = \boldsymbol{r}_0^*$
3. Do $k = 0, \ldots$ until convergence
4. $\quad \alpha_k = \boldsymbol{r}_k^{*T} \boldsymbol{r}_k / \boldsymbol{r}_k^{*T} A \boldsymbol{p}_k$
5. $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
6. $\quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k$
7. $\quad \boldsymbol{r}_{k+1}^* = \boldsymbol{r}_k^* - \alpha_k A^T \boldsymbol{p}_k^*$
8. $\quad \beta_k = \boldsymbol{r}_{k+1}^{*T} \boldsymbol{r}_{k+1} / \boldsymbol{r}_k^{*T} \boldsymbol{r}_k$
9. $\quad \boldsymbol{p}_k = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$
10. $\quad \boldsymbol{p}_k^* = \boldsymbol{r}_{k+1}^* + \beta_k \boldsymbol{p}_k^*$
11. End Do

If a dual system has to be solved, it is possible to compute its solution simply adding to Algorithm 11 the almost inexpensive recurrent relation:

$$\boldsymbol{x}_{k+1}^* = \boldsymbol{x}_k^* + \alpha_k \boldsymbol{p}_k^* \tag{181}$$

after computing the initial dual residual $\boldsymbol{r}_0^* = \boldsymbol{b}^* - A^T \boldsymbol{x}_0^*$. When, as is typically the case, a dual system does not exist the computation of the dual vectors is practically wasted and the solution to the native system by the Bi-CG algorithm is twice as expensive as that obtained with CG for a symmetric matrix.

3.4.2 Bi-Conjugate Gradient Stabilized (Bi-CGStab)

The Bi-CG algorithm is not much used in practice. Its main drawbacks are twofold:

1. each step needs the implementation of the product of A^T by a vector. This can be particularly expensive when the explicit form of A is not available. Moreover, the vectors computed with the use of A^T do not contribute directly to the solution, but are required only for the scalar coefficients α_k and β_k;

2. the convergence of Bi-CG proves typically quite erratic, with large oscillations of the residual vector norm. With ill-conditioned matrices this may lead to a breakdown due to the accumulation of rounding errors even though the underlying Lanczos biorthogonalization algorithm does not abort.

The reason for the erratic Bi-CG convergence relies on the way the scalar coefficients α_k and β_k are computed. Both factors need the inner product:

$$\rho_k = \boldsymbol{r}_k^{*T} \boldsymbol{r}_k \tag{182}$$

Recalling equation (170), the residual vector at step k can be written as the product of a polynomial of A with degree k by the initial residual $\boldsymbol{r}_0$:

$$\boldsymbol{r}_k = \phi_k(A)\, \boldsymbol{r}_0 \tag{183}$$

As $\boldsymbol{r}_k^*$ is computed with the same recurrence using A^T in place of A it follows that:

$$\boldsymbol{r}_k^* = \phi_k\left(A^T\right)\boldsymbol{r}_0^* \tag{184}$$

Thus, the inner product ρ_k is actually equal to:

$$\rho_k = \boldsymbol{r}_0^{*T}\left[\phi_k\left(A^T\right)\right]^T\phi_k\left(A\right)\boldsymbol{r}_0 = \boldsymbol{r}_0^{*T}\phi_k^2\left(A\right)\boldsymbol{r}_0 \tag{185}$$

Since the polynomial ϕ_k is squared the rounding errors in the computation of $\boldsymbol{r}_k$ and $\boldsymbol{r}_k^*$ are more damaging. In particular, high oscillations of the residual norm can cause the Bi-CG algorithm to become inaccurate.

A way to cope with such difficulties relies on stabilizing the erratic behaviour of the squared polynomial $\phi_k^2(A)$ replacing it with the product $\psi_k(A)\phi_k(A)$, where ψ_k is another polynomial with the aim of smoothing the oscillations of the residual norm. Define $\psi_k(A)$ as:

$$\psi_k\left(A\right) = \prod_{j=0}^{k-1}\left(I - \omega_j A\right) = \left(I - \omega_{k-1}A\right)\psi_{k-1}\left(A\right) \tag{186}$$

and a new smoothed $\boldsymbol{r}_k$ as:

$$\boldsymbol{r}_k = \psi_k\left(A\right)\phi_k\left(A\right)\boldsymbol{r}_0 \tag{187}$$

The current scalar ω is chosen so as to minimize the norm of the new smoothed residual vector. Note that also the vector $\boldsymbol{p}_k$ can be written as the product of another polynomial $\pi_k(A)$ by $\boldsymbol{r}_0$. Recalling the standard recursion for $\boldsymbol{r}_k$ in Bi-CG and equation (183), the new vector $\boldsymbol{r}_{k+1}$ according to (187) reads:

$$\boldsymbol{r}_{k+1} = \left(I - \omega_k A\right)\psi_k\left(A\right)\left[\phi_k\left(A\right)\boldsymbol{r}_0 - \alpha_k A\pi_k\left(A\right)\boldsymbol{r}_0\right] = \left(I - \omega_k A\right)\boldsymbol{s}_k \tag{188}$$

where:

$$\boldsymbol{s}_k = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k \tag{189}$$

and $\boldsymbol{p}_k$ is defined, similarly to $\boldsymbol{r}_k$, as the smoothed corresponding Bi-CG vector:

$$\boldsymbol{p}_k = \psi_k\left(A\right)\pi_k\left(A\right)\boldsymbol{r}_0 \tag{190}$$

Let us minimize $\|\boldsymbol{r}_{k+1}\|_2$ as a function of ω_k only. Using equation (188) it follows that:

$$\omega_k = \frac{\boldsymbol{s}_k^T A\boldsymbol{s}_k}{\left(A\boldsymbol{s}_k\right)^T A\boldsymbol{s}_k} \tag{191}$$

Equation (188), with (189), can be re-written as:

$$\boldsymbol{r}_{k+1} = \boldsymbol{s}_k - \omega_k A\boldsymbol{s}_k = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k - \omega_k A\boldsymbol{s}_k \tag{192}$$

Recalling that $\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$, equation (192) allows also for the update of the current solution $\boldsymbol{x}_{k+1}$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k + \omega_k\boldsymbol{s}_k \tag{193}$$

We need also the recurrence for the smoothed $\boldsymbol{p}_k$ defined in (190). Similarly to equation (188), the new $\boldsymbol{p}_{k+1}$ reads:

$$\boldsymbol{p}_{k+1} = (I - \omega_k A)\,\psi_k(A)\,[\phi_{k+1}(A)\,\boldsymbol{r}_0 + \beta_k \pi_k(A)\,\boldsymbol{r}_0] = \boldsymbol{r}_{k+1} + \beta_k (I - \omega_k A)\,\boldsymbol{p}_k \tag{194}$$

Finally, we need the recurrences for updating β_k and α_k. Both scalars use the inner product ρ_k defined in (185) which is not directly available as we avoid using squared polynomials. Instead we can easily compute $\tilde{\rho}_k$ as:

$$\tilde{\rho}_k = \boldsymbol{r}_0^{*T} \psi_k(A)\,\phi_k(A)\,\boldsymbol{r}_0 = \left[\psi_k\left(A^T\right)\boldsymbol{r}_0^*\right]^T \phi_k(A)\,\boldsymbol{r}_0 = \boldsymbol{r}_0^{*T}\boldsymbol{r}_k \tag{195}$$

To relate ρ_k with $\tilde{\rho}_k$ write the polymonials $\psi_k\left(A^T\right)$ and $\phi_k(A)$ as:

$$\psi_k\left(A^T\right) = \sum_{j=0}^{k} \eta_j^{(k)} \left(A^T\right)^{k-j} \tag{196}$$

$$\phi_k(A) = \sum_{j=0}^{k} \gamma_j^{(k)} A^{k-j} \tag{197}$$

and observe that because of the biorthogonalization property $\phi_k(A)\boldsymbol{r}_0$ is orthogonal to all vectors $\left(A^T\right)^i \boldsymbol{r}_0$ with $i < k$. Hence $\tilde{\rho}_k$ simplifies into:

$$\tilde{\rho}_k = \eta_0^{(k)} \boldsymbol{r}_0^{*T} A^k \phi_k(A)\,\boldsymbol{r}_0 \tag{198}$$

Now multiply and divide equation (198) by $\gamma_0^{(k)}$ and note that for increasing values of k the polynomial ϕ_k can be well approximated by its leading term, i.e. $\phi_k(A) \simeq \gamma_0^{(k)} A^k \boldsymbol{r}_0$. Hence $\tilde{\rho}_k$ reads:

$$\tilde{\rho}_k = \frac{\eta_0^{(k)}}{\gamma_0^{(k)}} \boldsymbol{r}_0^{*T} \phi_k^2(A)\,\boldsymbol{r}_0 = \frac{\eta_0^{(k)}}{\gamma_0^{(k)}} \rho_k \tag{199}$$

Recall the recurrent definition (186) of ψ_k. The leading coefficient $\eta_0^{(k+1)}$ can be inferred from this relation, obtaining:

$$\eta_0^{(k+1)} = -\omega_k \eta_0^{(k)} \tag{200}$$

The polynomial ϕ_k can be also defined by a short-term recurrence. Using the standard relations for $\boldsymbol{r}_k$ and $\boldsymbol{p}_k$ of Bi-CG yields:

$$\phi_{k+1}(A) = \phi_k(A) - \alpha_k A \pi_k(A) = (I - \alpha_k A)\,\phi_k(A) - \alpha_k \beta_k A \pi_{k-1}(A) \tag{201}$$

Hence the leading coefficient $\gamma_0^{(k+1)}$ reads:

$$\gamma_0^{(k+1)} = -\alpha_k \gamma_0^{(k)} \tag{202}$$

Finally combining equations (199) with (200) and (202) gives the scalar coefficient β_k:

$$\beta_k = \frac{\rho_{k+1}}{\rho_k} = \frac{\gamma_0^{(k+1)}}{\eta_0^{(k+1)}} \frac{\eta_0^{(k)}}{\gamma_0^{(k)}} \frac{\tilde{\rho}_{k+1}}{\tilde{\rho}_k} = \frac{\tilde{\rho}_{k+1}}{\tilde{\rho}_k} \frac{\alpha_k}{\omega_k} \tag{203}$$

The scalar α_k can be deduced in a simpler way. By its definition we can write:

$$\alpha_k = \frac{\boldsymbol{r}_0^{*T} \phi_k^2 (A) \boldsymbol{r}_0}{\boldsymbol{r}_0^{*T} \phi_k (A) A\pi_k (A) \boldsymbol{r}_0} \tag{204}$$

Now we can replace ϕ_k at both the numerator and the denominator with ψ_k. As α_k is a ratio, its value does not change obtaining:

$$\alpha_k = \frac{\boldsymbol{r}_0^{*T} \psi_k (A) \phi_k (A) \boldsymbol{r}_0}{\boldsymbol{r}_0^{*T} A\psi_k (A) \pi_k (A) \boldsymbol{r}_0} = \frac{\tilde{\rho}_k}{\boldsymbol{r}_0^{*T} A\boldsymbol{p}_k} \tag{205}$$

Putting all these relationships together we get the *Bi-Conjugate Gradient Stabilized* (Bi-CGStab) algorithm developed by van der Vorst in 1992:

ALGORITHM 12: BI-CONJUGATE GRADIENT STABILIZED

1. Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$ and set $\boldsymbol{r}_0^* = \boldsymbol{r}_0$
2. Set $\boldsymbol{p}_0 = \boldsymbol{r}_0$
3. DO $k = 0, \ldots$ until convergence
4. $\quad \alpha_k = \tilde{\rho}_k / \boldsymbol{r}_0^{*T} A\boldsymbol{p}_k$
5. $\quad \boldsymbol{s}_k = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k$
6. $\quad \omega_k = \boldsymbol{s}_k^T A\boldsymbol{s}_k / (A\boldsymbol{s}_k)^T A\boldsymbol{s}_k$
7. $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k + \omega_k \boldsymbol{s}_k$
8. $\quad \boldsymbol{r}_{k+1} = \boldsymbol{s}_k - \omega_k A\boldsymbol{s}_k$
9. $\quad \tilde{\rho}_{k+1} = \boldsymbol{r}_0^{*T} \boldsymbol{r}_{k+1}$
10. $\quad \beta_k = \tilde{\rho}_{k+1} \alpha_k / \tilde{\rho}_k \omega_k$
11. $\quad \boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k - \omega_k \beta_k A\boldsymbol{p}_k$
12. END DO

Notice that the Bi-CGStab algorithm succeeds in the twofold goal of stabilizing the erratic Bi-CG convergence and avoiding the use of A^T. This is why it is often referred to as a transpose-free variant of Bi-CG. The cost per iteration is practically the same as Bi-CG involving two matrix-vector products, but its smoother convergence makes it preferable also from a computational viewpoint. Finally observe that, despite its non trivial derivation, Bi-CGStab is very easy to implement also in a parallel environment, involving only short-term recurrences.

3.4.3 Quasi-Minimal Residual (QMR) algorithms

The two-sided Lanczos algorithm has given rise to a second class of iterative methods. The main reason for developing a new class of algorithms stems from the fact that Bi-CG and Bi-CGStab do not ensure any minimization property for the current approximate solution, i.e., it could not be optimal in the current subspace. In other words, these methods may lack in robustness since convergence is not theoretically guaranteed.

The basic idea relies on minimizing the current residual norm as is done in the GMRES algorithm. Recalling equation (159) the current residual can be written as:

$$\boldsymbol{r}_m = \boldsymbol{r}_0 - AV_m \boldsymbol{z} = V_{m+1} \left(\beta \boldsymbol{i}_1 - \overline{T}_m \boldsymbol{z} \right) \tag{206}$$

where it has been assumed, as usual, that $\boldsymbol{v}_1 = \boldsymbol{r}_0/\beta$. The norm of the residual vector is therefore:

$$\|\boldsymbol{r}_m\|_2 = \|V_{m+1}\left(\beta \boldsymbol{r}_1 - \overline{T}_m \boldsymbol{z}\right)\|_2 \tag{207}$$

i.e., a quadratic function of $\boldsymbol{z}$ which can be minimized. Contrary to GMRES, however, the matrix V_{m+1} cannot be neglected in the minimal norm computation since its columns are no longer orthonormal. Nevertheless, it is still a reasonable idea to minimize the function:

$$J(\boldsymbol{z}) = \|\beta \boldsymbol{i}_1 - \overline{T}_m \boldsymbol{z}\|_2 \tag{208}$$

and compute the corresponding approximate solution $\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m \boldsymbol{z}$. Such an approach is denoted as *Quasi-Minimal Residual* (QMR) and was introduced by Freund and Nachtigale in 1994. As the matrix $\overline{T}_m$ is tridiagonal, the solution to the resulting least-square problem is much less expensive than in the GMRES algorithm, thus providing some computational advantages. Moreover, using at each step a QR factorization of $\overline{T}_m$ to minimize $J(\boldsymbol{z})$ it is also possible to derive a short-term recurrence with a limited requirement of core memory. Even though the residual norm is not actually minimized at each step, hence the current approximate solution is not optimal in $\mathcal{K}_m(A, \boldsymbol{r}_0)$, the experience shows that generally the quasi-residual norm turns out to be close to the actual residual norm.

Among the QMR algorithms the transpose-free variant TFQMR has had some success in the applications. The variant *Symmetric Quasi-Minimal Residual* (SQMR) for symmetric indefinite matrices is even more successful. The key idea for SQMR is that it is possible to prove that the sequence of vectors $\boldsymbol{w}_i$ can also be computed recursively from the $\boldsymbol{v}_i$, with a significant acceleration of the Lanczos algorithm.