Acta Numerica (2025), pp. 805–890
doi:10.1017/S0962492924000096

Acceleration methods for fixed-point iterations

Yousef Saad
Department of Compter Science and Engineering, University of Minnesota,
Twin cities, MN 55455, USA
E-mail: saad@umn.edu

A pervasive approach in scientific computing is to express the solution to a given problem as the limit of a sequence of vectors or other mathematical objects. In many situations these sequences are generated by slowly converging iterative procedures, and this led practitioners to seek faster alternatives to reach the limit. 'Acceleration techniques' comprise a broad array of methods specifically designed with this goal in mind. They started as a means of improving the convergence of general scalar sequences by various forms of 'extrapolation to the limit', i.e. by extrapolating the most recent iterates to the limit via linear combinations. Extrapolation methods of this type, the best-known of which is Aitken's delta-squared process, require only the sequence of vectors as input.

However, limiting methods to use only the iterates is too restrictive. Accelerating sequences generated by fixed-point iterations by utilizing both the iterates and the fixed-point mapping itself has proved highly successful across various areas of physics. A notable example of these fixed-point accelerators (FP-accelerators) is a method developed by Donald Anderson in 1965 and now widely known as Anderson acceleration (AA). Furthermore, quasi-Newton and inexact Newton methods can also be placed in this category since they can be invoked to find limits of fixed-point iteration sequences by employing exactly the same ingredients as those of the FP-accelerators.

This paper presents an overview of these methods – with an emphasis on those, such as AA, that are geared toward accelerating fixed-point iterations. We will navigate through existing variants of accelerators, their implementations and their applications, to unravel the close connections between them. These connections were often not recognized by the originators of certain methods, who sometimes stumbled on slight variations of already established ideas. Furthermore, even though new accelerators were invented in different corners of science, the underlying principles behind them are strikingly similar or identical.

The plan of this article will approximately follow the historical trajectory of extrapolation and acceleration methods, beginning with a brief description of extrapolation ideas, followed by the special case of linear systems, the application to self-consistent field (SCF) iterations, and a detailed view of Anderson acceleration. The last part of the paper is concerned with more recent developments, including theoretical aspects, and a few thoughts on accelerating machine learning algorithms.

2020 Mathematics Subject Classification: 65B05, 65B99, 65F10, 65H10

CONTENTS

1. Historical perspective and overview

Early iterative methods for solving systems of equations, whether linear or nonlinear, often relied on simple *fixed-point iterations* of the form

$$x_{j+1} = g(x_j), \tag{1.1}$$

which, under certain conditions, converge to a *fixed point* x_* of g, i.e. a point such that $g(x_*) = x_*$. Here, g is some mapping from $\mathbb{R}^n$ to itself which we assume to be at least continuous. Thus the iterative method for solving linear systems originally developed by Gauss in 1823 and commonly known today as the Gauss–Seidel iteration (see e.g. Saad 2003) can be recast in this form. The fixed-point iterative approach can be trivially adopted for solving a system of equations of the form

$$f(x) = 0, \tag{1.2}$$

where f is again a mapping from $\mathbb{R}^n$ to itself. This can be achieved by selecting a non-zero scalar γ and defining the mapping $g(x) \equiv x + \gamma f(x)$, whose fixed points are identical to the zeros of f. The process would generate the iterates

$$x_{j+1} = x_j + \gamma f(x_j), \quad j = 0, 1, \ldots \tag{1.3}$$

starting from some initial guess x_0. Approaches that utilize the above framework are common in optimization, where $f(x)$ is the negative of the gradient of a certain objective function $\phi(x)$ whose minimum is sought. The simplicity and versatility of the fixed-point iteration method for solving nonlinear equations are among the reasons it has been successfully used across various fields.

Sequences of vectors, whether of the type (1.1) introduced above or generated by some other 'black-box' process, often converge slowly or may even fail to converge. As a result practitioners have long sought to build sequences that converge faster to the same limit as the original sequence, or even to establish convergence in case the original sequence fails to converge. We need to emphasize a key distinction between two different strategies that have been adopted in this context. In the first, we are just given all or a few of the recent members of the sequence and no other

information, and the goal is to produce another sequence from it, that will hopefully converge faster. These procedures typically rely on forming a linear combination of the current iterate with previous ones in an effort to essentially extrapolate to the limit, and for this reason they are often called *extrapolation methods*. Starting from a sequence $\{x_j\}_{j=0,1,\ldots}$, a typical extrapolation technique builds the new, extrapolated, sequence as follows:

$$y_j = \sum_{i=[j-m]}^{j} \gamma_i^{(j)} x_i, \tag{1.4}$$

where we recall the notation $[j-m] = \max\{j-m, 0\}$ and m is a parameter, known as the 'depth' or 'window size', and the $\gamma_i^{(j)}$ are 'acceleration parameters'. If the new sequence is to converge to the same limit as the original one, the condition

$$\sum_{i=[j-m]}^{j} \gamma_i^{(j)} = 1 \tag{1.5}$$

must be imposed. Because the process works by forming linear combinations of iterates of the original sequences, it is also often termed a *linear acceleration* procedure (see e.g. Brezinski and Redivo-Zaglia 1991, Brezinski 2000, Forsythe 1953), but the term 'extrapolation' is more common. Extrapolation methods are discussed in Section 2.

In the second strategy we are again given all or a few of the recent iterates, but now we also have access to the fixed-point mapping g to help build the next member of the sequence. A typical example of these methods is the Anderson/Pulay acceleration, which is discussed in detail in Sections 4.3 and 6. When presented from the equivalent form of Pulay mixing, this method builds a new iterate as follows:

$$x_{j+1} = \sum_{i=[j-m]}^{j} \theta_i^{(j)} g(x_i), \tag{1.6}$$

where the $\theta_i^{(j)}$ satisfy the constraint $\sum_{i=[j-m]}^{j} \theta_i^{(j)} = 1$. As can be seen, the function g is now invoked when building a new iterate. Of course, having access to g may allow us to develop more powerful methods than if we were to use only the iterates, as is done in extrapolation methods. It is also clear that there are situations when this is not practically feasible.

The emphasis of this paper is on this second class, which we will refer to as *fixed-point acceleration methods*, or *FP-acceleration methods*. Due to their broad applicability, FP-acceleration emerged from different corners of science. Ideas related to extrapolation can be traced as far back as the seventeenth century; see Brezinski and Redivo-Zaglia (2019, § 6) for references. However, one can say that modern extrapolation and acceleration ideas took off with the work of Richardson (1910) and Aitken (1926), among others, in the early part of the twentieth century

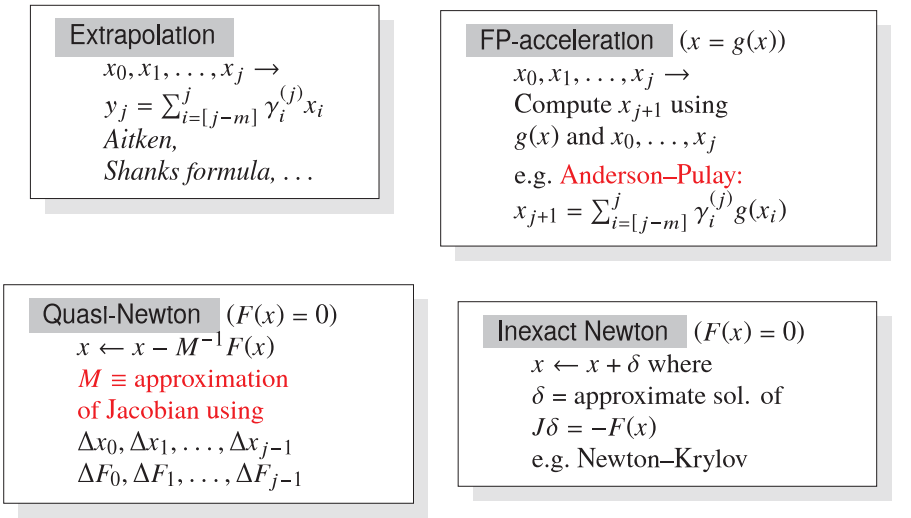


Figure 1.1. Four distinct classes of acceleration methods.

and then gained importance toward the mid-1950s, with the advent of electronic computers; e.g. Romberg (1955) and Shanks (1955).

Because acceleration and extrapolation methods are often invoked for solving nonlinear equations, they naturally compete with 'second-order type methods' such as those based on quasi-Newton approaches. Standard Newton-type techniques cannot be placed in the same category of methods as those described above because they exploit a local second-order model and invoke the differential of f explicitly. In many practical problems, obtaining the Jacobian of f is not feasible, or it may be too expensive. However, inexact Newton and quasi-Newton methods are two alternatives that bypass the need to compute the whole Jacobian J of f. Therefore they satisfy our requirements for what can be viewed as a method for accelerating iterations of the form (1.3), in that they only invoke previous iterates and our ability to apply the mapping g (or f) to any given vector. It should therefore come as no surprise that one can find many interesting connections between this class of methods and some of the FP-acceleration techniques. A few of the methods developed in the quasi-Newton context bear strong similarities, and are in some cases even mathematically equivalent, to techniques from the FP-acceleration class.

The four classes of methods discussed above are illustrated in Figure 1.1. This article aims to provide coverage of these four classes of acceleration and interpolation methods, with an emphasis on those geared toward accelerating fixed-point iterations, i.e. those in the top-right corner of Figure 1.1. There is a vast amount of literature on these methods, and it would be challenging and unrealistic to try to

be exhaustive. However, one of our specific goals is to unravel the various connections between the different methods. Another is to present, whenever possible, interesting variants of these methods and details of their implementations.

Notation

- Throughout the article, g will denote a mapping from $\mathbb{R}^n$ to itself, and we are interested in a fixed point x_* of g. Similarly, f will also denote a mapping from $\mathbb{R}^n$ to itself, and we are interested in a zero of f.
- $\{x_j\}_{j=0,1,\ldots}$ denotes a sequence in $\mathbb{R}^n$.
- Given a sequence $\{x_j\}_{j=0,1,\ldots}$, the forward difference operator Δ builds a new sequence whose terms are defined by
$$\Delta x_j = x_{j+1} - x_j, \quad j = 0, 1, \ldots. \tag{1.7}$$
- We will often refer to an evolving set of columns where the most recent m vectors from a sequence are retained. In order to cope with the different indexing used in the algorithms, we found it convenient to define, for any $k \in \mathbb{Z}$,
$$[k] = \max\{k, 0\}. \tag{1.8}$$
Thus we will often see matrices of the form
$$X_j = [x_{[j-m+1]}, x_{[j-m+1]+1}, \ldots, x_j]$$
that have $j + 1$ columns when $j < m$, and m columns otherwise.
- Throughout the paper, $\|\cdot\|_2$ will denote the Euclidean norm and $\|\cdot\|_F$ is the Frobenius norm on matrices.

2. Extrapolation methods for general sequences

Given a sequence $\{x_j\}_{j=0,1,\ldots}$, an extrapolation method builds an auxiliary sequence $\{y_j\}_{j=0,1,\ldots}$, where y_j is typically a linear combination of the most recent iterates as in (1.4). The goal is to produce a sequence that converges faster to the limit of x_j. Here m is a parameter known as the 'depth' or 'window size', and the $\gamma_i^{(j)}$ are 'acceleration parameters', which sum up to one; see (1.5). Aitken's delta-squared process (Aitken 1926) is an early instance of such a procedure that had a major impact.

2.1. Aitken's procedure

Suppose we have a scalar sequence $\{x_i\}$ for $i = 0, 1, \ldots$ that converges to a limit x_*. Aitken assumed that in this situation the sequence roughly satisfies the relation
$$x_{j+1} - x_* = \lambda(x_j - x_*) \quad \text{for all } j, \tag{2.1}$$

where λ is some unknown scalar. This is simply an expression of a geometric convergence to the limit. The above condition defines a set of sequences and the condition is termed a 'kernel'. In this particular case (2.1) is the Aitken kernel.

The scalar λ, and the limit x_* can be trivially determined from three consecutive iterates x_j, x_{j+1}, x_{j+2} by writing

$$\frac{x_{j+1} - x_*}{x_j - x_*} = \lambda, \quad \frac{x_{j+2} - x_*}{x_{j+1} - x_*} = \lambda, \tag{2.2}$$

from which it follows by eliminating λ from the two equations that

$$x_* = \frac{x_j x_{j+2} - x_{j+1}^2}{x_{j+2} - 2x_{j+1} + x_j} = x_j - \frac{(\Delta x_j)^2}{\Delta^2 x_j}. \tag{2.3}$$

Here Δ is the forward difference operator defined earlier in (1.7), and $\Delta^2 x_j = \Delta(\Delta x_j)$. As can be expected, the set of sequences that satisfy Aitken's kernel, i.e. (2.1), is very narrow. In fact, subtracting the same relation obtained by replacing j by $j-1$ from relation (2.1), we would obtain $x_{j+1} - x_j = \lambda(x_j - x_{j-1})$, hence $x_{j+1} - x_j = \lambda^j(x_1 - x_0)$. Therefore scalar sequences that satisfy Aitken's kernel exactly are of the form

$$x_{j+1} = x_0 + (x_1 - x_0) \sum_{k=0}^{j} \lambda^k \quad \text{for } j \geq 0. \tag{2.4}$$

Although a given sequence is unlikely to satisfy Aitken's kernel exactly, it may nearly satisfy it when approaching the limit, and in this case the extrapolated value (2.3) will provide a way to build an 'extrapolated' sequence defined as follows:

$$y_j = x_j - \frac{(\Delta x_j)^2}{\Delta^2 x_j} \quad j = 0, 1, \ldots. \tag{2.5}$$

Note that to compute y_j we must have available three consecutive iterates, namely x_j, x_{j+1}, x_{j+2}, so y_j starts with a delay relative to the original sequence.

A related approach known as Steffensen's method is geared toward solving the equation $f(x) = 0$ by the Newton-like iteration

$$x_{j+1} = x_j - \frac{f(x_j)}{d(x_j)}, \quad \text{where } d(x) = \frac{f(x + f(x)) - f(x)}{f(x)}. \tag{2.6}$$

One can recognize in $d(x)$ an approximation of the derivative of f at x. This scheme converges quadratically under some smoothness assumptions for f. In addition, it can be easily verified that when the sequence $\{x_j\}$ is produced from the fixed-point iteration $x_{j+1} = g(x_j)$, one can recover Aitken's iteration for this sequence by applying Steffensen's scheme to the function $f(x) = g(x) - x$. Therefore Aitken's method also converges quadratically for such sequences when g is smooth enough.

2.2. *Generalization: The Shanks transform and the ϵ-algorithm*

Building on the success of Aitken's acceleration procedure, Shanks (1955) explored ways to generalize it by replacing the kernel (2.1) with a kernel of the form

$$a_0(x_j - x_*) + a_1(x_{j+1} - x_*) + \cdots + a_m(x_{j+m} - x_*) = 0, \tag{2.7}$$

where the scalars $a_0, \ldots, a_m$ and the limit x_* are unknowns. The sum of the scalars a_i cannot be equal to 0 and there is no loss of generality in assuming that they add up to 1, that is,

$$a_0 + a_1 + \cdots + a_m = 1. \tag{2.8}$$

In addition, it is commonly assumed that $a_0 a_m \neq 0$, so that exactly $m+1$ terms are involved at any given step. We can set up a linear system to compute x_* by putting equation (2.7) and (2.8) together into the $(m+2) \times (m+2)$ linear system

$$\begin{cases} a_0 & + a_1 & + \cdots + a_m & & = 1, \\ a_0 x_{j+i} & + a_1 x_{j+i+1} & + \cdots + a_m x_{j+i+m} & - x_* & = 0, \quad i = 0, \ldots, m. \end{cases} \tag{2.9}$$

Cramer's rule can now be invoked to solve this system and derive a formula for x_*. With a few row manipulations, we end up with the following formula, known as *the Shanks (or 'Schmidt–Shanks') transformation for scalar sequences*:

$$y_j^{(m)} = \frac{\begin{vmatrix} x_j & x_{j+1} & \cdots & x_{j+m} \\ \Delta x_j & \Delta x_{j+1} & \cdots & \Delta x_{j+m} \\ \vdots & \vdots & \vdots & \vdots \\ \Delta x_{j+m-1} & \Delta x_{j+m} & \cdots & \Delta x_{j+2m-1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 \\ \Delta x_j & \Delta x_{j+1} & \cdots & \Delta x_{j+m} \\ \vdots & \vdots & \vdots & \vdots \\ \Delta x_{j+m-1} & \Delta x_{j+m} & \cdots & \Delta x_{j+2m-1} \end{vmatrix}}. \tag{2.10}$$

A more elegant way to derive the above formula – one that will lead to useful extensions – is to exploit the following relation, which follows from the kernel (2.7):

$$a_0 \Delta x_j + a_1 \Delta x_{j+1} + \cdots + a_m \Delta x_{j+m} = 0. \tag{2.11}$$

With this, we will build a new system, now for the unknowns $a_0, a_1, \ldots, a_m$, as follows:

$$\begin{cases} a_0 & + a_1 & + \cdots + a_m & = 1, \\ a_0 \Delta x_{j+i} & + a_1 \Delta x_{j+i+1} & + \cdots + a_m \Delta x_{j+i+m} & = 0, \quad i = 0, \ldots, m-1. \end{cases} \tag{2.12}$$

The right-hand side of this $(m+1)\times(m+1)$ system is the vector $e_1 = [1, 0, \ldots, 0]^\top \in \mathbb{R}^{m+1}$. Using Cramer's rule will yield an expression for a_k as the fraction of two determinants. The denominator of this fraction is the same as that of (2.10). The numerator is $(-1)^k$ times the determinant of that same denominator where the first

row and the $(k+1)$th column are deleted. Substituting these formulas for a_k into the sum $a_0 x_j + a_1 x_{j+1} + \cdots + a_m x_{j+m}$ will result in an expression that amounts to expanding the determinant in the numerator of (2.10) with respect to its first row. By setting $a_i \equiv \gamma_i^{(m)}$ we can therefore rewrite Shank's formula in the form

$$y_j^{(m)} = \gamma_0^{(m)} x_j + \gamma_1^{(m)} x_{j+1} + \cdots + \gamma_m^{(m)} x_{j+m}, \tag{2.13}$$

where each $\gamma_j^{(m)}$ is the ratio of two determinants, as was just explained.

It can be immediately seen that the particular case $m = 1$ yields exactly Aitken's delta-squared formula:

$$y_j^{(2)} = \frac{\begin{vmatrix} x_j & x_{j+1} \\ \Delta x_j & \Delta x_{j+1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ \Delta x_j & \Delta x_{j+1} \end{vmatrix}} = \frac{x_j \Delta x_{j+1} - x_{j+1} \Delta x_j}{\Delta^2 x_j} = x_j - \frac{(\Delta x_j)^2}{\Delta^2 x_j}.$$

As written, the above expressions are meant for scalar sequences, but they can be generalized to vectors and this will be discussed later.

The generalization discovered by Shanks relied on ratios of determinants of order m, where m is the depth of the recurrence defined above. The non-practical character of this technique prompted Peter Wynn (1956) to explore an alternative implementation, and this resulted in an amazingly simple formula which he dubbed the 'ϵ-algorithm'. This remarkable discovery spurred a huge following among the numerical linear algebra community, e.g. Cabay and Jackson (1976), Eddy and Wang (1979), Brezinski (1980), Eddy (1979), Mešina (1977), Sidi, Ford and Smith (1986), Wynn (1962), Brezinski (1977), Kaniel and Stein (1974), Brezinski (1975), Jbilou (1988), Jbilou and Sadok (1991) and Germain-Bonne (1978), among many others. Brezinski (2000) gives a rather exhaustive review of these techniques up to the year 2000. More recently, Brezinski and Redivo-Zaglia (2019) surveyed these methods while also providing a wealth of information on the history of their development.

So what is Wynn's procedure to compute the $y_j^{(m)}$? The formula given above leads to expressions with determinants that involve Hankel matrices from which some recurrences can be obtained, but these are not only complicated but also numerically unreliable.

Wynn's ϵ-algorithm is a recurrence relation to compute $y_j^{(m)}$ shown in equation (2.14). It defines sequences $\{\epsilon_j^{(m)}\}_{j=0,1,\ldots}$, each indexed by an integer[1] m starting with $m = -1$.

[1] Note that this article adopts a different notation from the common usage in the literature: the index of the sequence is a subscript rather than a superscript. In Wynn's notation $\{\epsilon_m^{(j)}\}_{j=0,1,\ldots}$ denotes the mth extrapolated sequence.

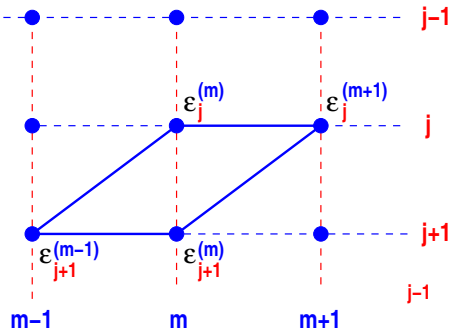


Figure 2.1. Wynn's rhombus rule for the ϵ-algorithm.

The sequence $\{\epsilon_j^{(-1)}\}$ is just the zero sequence $\epsilon_j^{(-1)} = 0$, for all j, while the sequence $\{\epsilon_j^{(0)}\}$ is the original sequence $\epsilon_j^{(0)} = x_j$, for all j. The other sequences, i.e. $\{\epsilon_j^{(m)}\}$, are then obtained recursively as follows:

$$\epsilon_j^{(-1)} = 0, \quad \epsilon_j^{(0)} = x_j, \quad \epsilon_j^{(m+1)} = \epsilon_{j+1}^{(m-1)} + \frac{1}{\epsilon_{j+1}^{(m)} - \epsilon_j^{(m)}} \quad \text{for } m, j \geq 0. \tag{2.14}$$

Each sequence $\{\epsilon_j^{(m)}\}$, where m is constant, can be placed on a vertical line on a grid, as shown in Figure 2.1. With this representation, the jth iterate for the $(m+1)$th sequence can be obtained using a simple rhombus rule from two members of the mth sequence and one member of the $(m-1)$th sequence, as is shown in the figure. Only the even-numbered sequences are accelerations of the original sequence. The odd-numbered ones are just intermediate auxiliary sequences. For example, the sequence $\epsilon_2^{(m)}$ is identical to the sequence obtained from Aitken's delta-squared process but the sequence $\epsilon_1^{(m)}$ is auxiliary.

If we let η_m denote the transformation that maps a sequence $\{x_j\}$ into $\{t_j^{(m)}\}$, i.e. we have $\eta_m(x_j) = t_j^{(m)}$, then it can be shown that $\epsilon_j^{(2m)} = \eta_m(x_j)$ and $\epsilon_j^{(2m+1)} = 1/\eta_m(\Delta x_j)$.

From a computational point of view, we would proceed differently in order not to have to store all the sequences. The process, shown in Figure 2.2, works on diagonals of the table. When the original sequence is computed we usually generate $x_0, x_1, \ldots, x_j, \ldots$ in this order. Assuming that we need to compute up to the mth sequence $\{\epsilon_j^{(m)}\}$, then once $\epsilon_j^{(0)} = x_j$ becomes available we can obtain the entries $\epsilon_{j-1}^{(1)}, \epsilon_{j-2}^{(2)}, \ldots, \epsilon_{j-m}^{(m)}$. This will require the entries $\epsilon_{j-1}^{(0)}, \epsilon_{j-2}^{(1)}, \ldots, \epsilon_{j-m}^{(m-1)}$, so we will only need to keep the previous diagonal.

Wynn's result on the equivalence between the ϵ-algorithm and Shank's formula is stunning not only for its elegance but also because it is highly non-trivial and complex to establish; see comments regarding this in Brezinski (1980, p. 162).

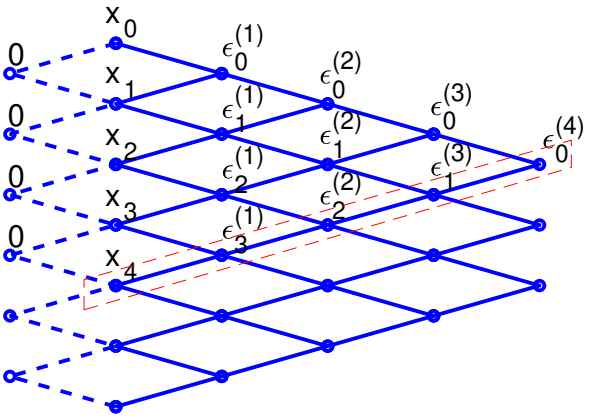


Figure 2.2. Computational diagram of the ϵ algorithm.

2.3. *Numerical illustration*

Extrapolation methods were quite popular for dealing with scalar sequences such as those originating from numerical integration, or from computing numerical series. The following illustration highlights the difference between fixed-point acceleration and extrapolation.

In the example we compute π from the Arctan expansion

$$\text{atan}(z) = z - \frac{z^3}{3} + \frac{z^5}{5} - \frac{z^7}{7} + \cdots = \sum_{j=0}^{\infty} \frac{(-1)^j z^{2j+1}}{2j+1}. \tag{2.15}$$

Applying this to $z = 1$ will give a sequence that converges to $\pi/4$. For an arbitrary z, we would take the following sequence, starting with $x_0 = 0$:

$$x_{j+1} = x_j + \frac{(-1)^j z^{2j+1}}{2j+1}, \quad j = 0, 1, \ldots, n-1. \tag{2.16}$$

In this illustration, we take $n = 30$ and $z = 1$.

Figure 2.3 shows the convergence of the original sequence along with four extrapolated sequences. In the figure, 'Aitken^2' refers to Aitken applied twice, that is, Aitken is applied to the extrapolated sequence obtained from applying the standard Aitken process. Note that all extrapolated sequences are shorter since, as was indicated above, a few iterates from the original sequence are needed in order to get the new sequence started. The plot shows a remarkable improvement in convergence relative to the original sequence: 15 digits of accuracy are obtained for the sixth-order ϵ-algorithm in 20 steps, in comparison to barely two digits obtained with 30 steps of the original sequence.

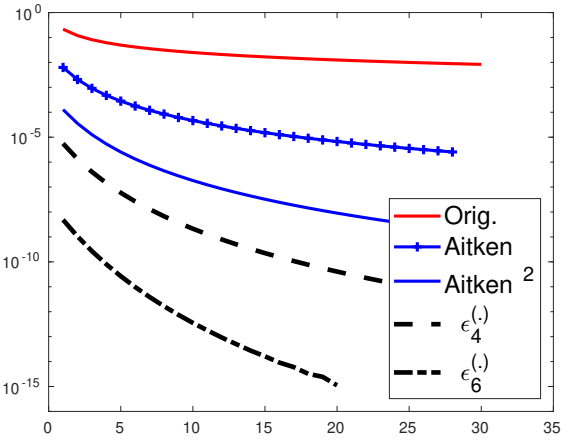


Figure 2.3. Comparison of a few extrapolation methods for computing $\pi/4$ with formula (2.16).

2.4. *Vector sequences*

Extrapolation methods have been generalized to vector sequences in a number of ways. There is no canonical generalization that seems compelling and natural, but the simplest consists of applying the acceleration procedure componentwise. However, this naive approach is not recommended in the literature as it fails to capture the intrinsic vector character of the sequence. Many of the generalizations relied on extending in some way the notion of inverse of a vector or that of the division of a vector by another vector. For the Aitken procedure this leads to quite a few possible generalizations; see e.g. Ramière and Helfer (2015).

Peter Wynn himself considered a generalization of his scheme to vectors (Wynn 1962). Note that generalizing the recurrence formula (2.14) of the ϵ-algorithm only requires that we define the inverse of a vector. To this end, Wynn considered several options and ended up adopting a definition proposed by Samelson:

$$x^{-1} = \frac{\bar{x}}{(\bar{x}, x)} = \frac{\bar{x}}{\|x\|_2^2}, \tag{2.17}$$

where $\bar{x}$ denotes the complex conjugate of x. This is simply the pseudo-inverse of x viewed as a matrix from $\mathbb{C}^1$ to $\mathbb{C}^n$. It is also known as the Samelson inverse of a vector. Recently, Brezinski, Redivo-Zaglia and Salam (2023) considered more complex extensions of the ϵ-algorithm by resorting to Clifford algebra.

The various generalizations of the ϵ-algorithm to vectors based on extending the inverse of a vector were not too convincing to practitioners. For this reason, Brezinski (1977, 1975) adopted a different approach that essentially introduced duality as a tool. We start with the simple case of Aitken's acceleration, for which

we rewrite the original ansatz (2.1) as follows:

$$x_* = x_j + \mu \Delta x_j,$$

where $\mu = 1/(1-\lambda)$ is a scalar. In the scalar case, it can be readily seen from (2.3) that μ is given by

$$\mu = -\frac{\Delta x_j}{\Delta^2 x_j}. \tag{2.18}$$

In the vector case, we need μ to be a scalar, and so a natural extension to vector sequences would entail taking inner products of the numerator and denominator in (2.18) with a certain vector w:

$$\mu = -\frac{(w, \Delta x_j)}{(w, \Delta^2 x_j)}. \tag{2.19}$$

This leads to the formula

$$y_j = x_j - \frac{(w, \Delta x_j)}{(w, \Delta^2 x_j)}\, \Delta x_j = \frac{\begin{vmatrix} x_j & x_{j+1} \\ (w, \Delta x_j) & (w, \Delta x_{j+1}) \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ (w, \Delta x_j) & (w, \Delta x_{j+1)} \end{vmatrix}}. \tag{2.20}$$

We need to clarify notation. The determinant in the numerator of the right-hand side in (2.20) now has the vectors x_j and x_{j+1} in its first row, instead of scalars. This is to be interpreted with the help of the usual expansion of this determinant with respect to this row. Thus this determinant is evaluated as

$$\begin{aligned}(w, \Delta x_{j+1})x_j - (w, \Delta x_j)x_{j+1} &= (w, \Delta x_{j+1} - \Delta x_j)x_j + (w, \Delta x_j)(x_j - x_{j+1}) \\ &= (w, \Delta^2 x_j)x_j - (w, \Delta x_j)\Delta x_j.\end{aligned}$$

This establishes the second equality in (2.20) by noting that the denominator in the last expression of (2.20) is just $(w, \Delta^2 x_j)$.

The vector w can be selected in a number of ways, but it is rather natural to take $w = \Delta^2 x_j$, and this leads to

$$y_j = x_j - \frac{(\Delta x_j, \Delta^2 x_j)}{\|\Delta^2 x_j\|_2^2} \Delta x_j. \tag{2.21}$$

Our aim in showing the expression on the right of (2.20) is to unravel possible extensions of the more general Shanks formula (2.10). It turns out that the above formulation of Aitken acceleration for vector sequences is a one-dimensional version of the RRE extrapolation method; see Section 2.6 for details.

The vector version of Aitken's extrapolation described above can easily be extended to a vector form of the Shanks formula (2.10) by selecting a vector w and replacing every difference Δx_{j+i} in (2.10) by $(w, \Delta x_{j+i})$. The first row of the determinant in the numerator will still have the vectors x_{j+i} and the resulting determinant

is to be interpreted, as was explained above. Doing this would lead to the following extension of the Shanks formula (2.10):

$$y_j^{(m)} = \frac{\begin{vmatrix} x_j & x_{j+1} & \cdots & x_{j+m} \\ (w, \Delta x_j) & (w, \Delta x_{j+1}) & \cdots & (w, \Delta x_{j+m}) \\ \vdots & \vdots & \vdots & \vdots \\ (w, \Delta x_{j+m-1}) & (w, \Delta x_{j+m}) & \cdots & (w, \Delta x_{j+2m-1}) \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 \\ (w, \Delta x_j) & (w, \Delta x_{j+1}) & \cdots & (w, \Delta x_{j+m}) \\ \vdots & \vdots & \vdots & \vdots \\ (w, \Delta x_{j+m-1}) & (w, \Delta x_{j+m}) & \cdots & (w, \Delta x_{j+2m-1}) \end{vmatrix}}. \tag{2.22}$$

This generalization was advocated in Brezinski (1975).

However, there is a missing step in our discussion so far: although we now have a generalized Shanks formula, we still need an effective way to evaluate the related expressions, hopefully with something similar to the ϵ-algorithm, that avoids determinants. This is not an issue for Aitken's procedure, which corresponds to the case $m = 2$ as the corresponding determinants are trivial. For the cases where m is larger, a different approach is required. This was examined by Brezinski (1975, 1977), who proposed a whole class of techniques referred to as *topological ϵ-algorithms*; see also Brezinski and Redivo-Zaglia (2019) and Brezinski (1980).

2.5. *The projection viewpoint and MMPE*

Another way to extend extrapolation methods to vector sequences is to invoke a projection approach. We return to equations (2.12), where the second equation is taken with $i = 0$, and eliminate the first equation by setting $a_0 = 1 - a_1 - a_2 - \cdots - a_m$, which leads to

$$\Delta x_j + \sum_{i=1}^{m} a_i (\Delta x_{j+i} - \Delta x_j) = 0. \tag{2.23}$$

Making use of the simple identity $u_{j+i} - u_j = \Delta u_{j+i-1} + \Delta u_{j+i-2} + \cdots + \Delta u_j$ will show that the term $\Delta x_{j+i} - \Delta x_j$ is the sum of the vectors $\Delta^2 x_k$ for $k = j$ to $k = j + i - 1$, and this leads to the following reformulation of the left-hand side of (2.23):

$$\begin{aligned} \Delta x_j + \sum_{i=1}^{m} a_i \sum_{k=j}^{j+i-1} \Delta^2 x_k &= \Delta x_j + \sum_{k=j}^{j+m-1} [a_{k-j+1} + a_{k-j+2} + \cdots + a_m] \Delta^2 x_k \\ &\equiv \Delta x_j + \sum_{k=j}^{j+m-1} \beta_{k-j+1} \Delta^2 x_k, \end{aligned}$$

where we set $\beta_{k-j+1} \equiv a_{k-j+1} + a_{k-j+2} + \cdots + a_m$. Thus equation (2.23) becomes

$$\Delta x_j + \sum_{i=1}^{m} \beta_i \Delta^2 x_{j+i-1} = 0, \tag{2.24}$$

where $\beta_i = a_i + a_{i+1} + \cdots + a_m$. It is convenient to define

$$\Delta X_j = [\Delta x_j, \Delta x_{j+1}, \ldots, \Delta x_{j+m-1}], \tag{2.25}$$

$$\Delta^2 X_j = [\Delta^2 x_j, \Delta x_{j+1}, \ldots, \Delta^2 x_{j+m-1}], \tag{2.26}$$

$$\beta = [\beta_1, \beta_2, \ldots, \beta_m]^\top. \tag{2.27}$$

With this notation the system (2.23) takes the matrix form

$$\Delta x_j + \Delta^2 X_j \, \beta = 0. \tag{2.28}$$

This is an over-determined system of equations with m unknowns. Taking a projection viewpoint, we can select a set of vectors $W \in \mathbb{R}^{n \times m}$ and extract a solution to the projected system

$$W^\top (\Delta x_j + \Delta^2 X_j \, \beta) = 0. \tag{2.29}$$

Assuming that the $m \times m$ matrix $W^\top [\Delta^2 X_j]$ is non-singular, then the accelerated iterate exists and is given by

$$y_j = x_j - \Delta X_j \, \beta, \quad \text{where } \beta = \left[W^\top \Delta^2 X_j\right]^{-1} (W^\top \Delta x_j). \tag{2.30}$$

A number of methods developed in Brezinski (1975) were of the type shown above with various choices of the set W. Among these, one approach in particular is worth mentioning due to its connection with other methods.

This approach starts with another natural extensions of (2.22), which is to use a different vector w for each of the rows of the determinants, but apply these to the same Δx_k on the same column:

$$y_j^{(m)} = \frac{\begin{vmatrix} x_j & x_{j+1} & \cdots & x_{j+m} \\ (w_1, \Delta x_j) & (w_1, \Delta x_{j+1}) & \cdots & (w_1, \Delta x_{j+m}) \\ \vdots & \vdots & \vdots & \vdots \\ (w_m, \Delta x_j) & (w_m, \Delta x_{j+1}) & \cdots & (w_m, \Delta x_{j+m}) \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 \\ (w_1, \Delta x_j) & (w_1, \Delta x_{j+1}) & \cdots & (w_m, \Delta x_{j+m}) \\ \vdots & \vdots & \vdots & \vdots \\ (w_m, \Delta x_j) & (w_m, \Delta x_{j+1}) & \cdots & (w_m, \Delta x_{j+m}) \end{vmatrix}}. \tag{2.31}$$

It was first discussed in Brezinski (1975), and essentially the same method was independently published in the Russian literature in Pugachëv (1977). The method was later rediscovered by Sidi *et al.* (1986), who gave it the name *modified minimal*

polynomial extrapolation (MMPE), by which it is commonly known today. Because there is some freedom in the selection of the set W, MMPE represents more than just one method. We will discuss it further in Section 2.6.

It is rather interesting that the above determinant can be expressed in the same form as (2.30). To see this, we need to process the numerator and denominator of (2.31) as follows. Starting from the last column and proceeding backwards, we subtract column $k-1$ from column k, and this is done for $k = m+1, m, \ldots, 2$. With this, the first row of the determinant in the denominator will be a one followed by m zeros. The first row of the determinant in the numerator will be the vector x_j followed by $\Delta x_j, \Delta x_{j+1}, \Delta x_{j+m-1}$. The entries in the first columns of both denominators are unchanged. The block consisting of entries $(2\colon m+1)\times(2\colon m+1)$ of both denominators will have the entries $(w_i, \Delta^2 x_{j+k-1})$ in its kth column, with $k = 1, \ldots, m$. This block is simply the matrix $W^\top \Delta^2 X_j$. To expand the resulting determinant in the numerator, we utilize the following relation, where τ is a scalar and S is an invertible $m \times m$ matrix:

$$\begin{vmatrix} \tau & f \\ b & S \end{vmatrix} = \det(S)[\tau - fS^{-1}b]. \tag{2.32}$$

This relation is also true in the case when τ is a vector in $\mathbb{R}^{\nu\times 1}$ and f is in $\mathbb{R}^{\nu\times m}$ with the interpretation of such determinants seen earlier. After transforming the numerator of (2.31) as discussed above, we will obtain a determinant in the form of the left-hand side of (2.32) in which $f = \Delta X_j$, $b = W^\top \Delta X_j$ and $S = W^\top \Delta^2 X_j$. Applying the above formula to this determinant results in the expression (2.30).

Vector acceleration algorithms were also defined as processes devoted solely to vector sequences generated from linear iterative procedures. The next section explores this framework a little further.

2.6. *RRE and related methods*

The difficulties encountered in extending Aitken's method and the ϵ-algorithm to vector sequences led researchers to seek better motivated alternatives, by focusing on vector sequences generated from specific linear processes. Thus Cabay and Jackson (1976) introduced a method called the minimal polynomial extrapolation (MPE) which exploits the low-rank character of the set of sequences in the linear case. At about the same time, Eddy (1979) and Mešina (1977) developed a method dubbed 'reduced rank extrapolation' (RRE), which was quite similar in spirit to MPE. Our goal here is mainly to link these methods with the ones seen earlier and developed from a different viewpoint.

We begin by describing RRE by following the notation and steps of the original paper by Mešina (1977). RRE was initially designed for vector sequences generated from the following fixed-point (linear) iterative process:

$$x_{j+1} = Mx_j + f. \tag{2.33}$$

Mešina chose to express the extrapolated sequence in the form

$$y_m = x_0 + \sum_{i=1}^{m} \beta_i \Delta x_{i-1}, \tag{2.34}$$

where β_i, $i = 1, \ldots, m$ are scalars to be determined.

With the notation (2.25)–(2.27) we can write $y_m = x_0 + \Delta X_0 \beta$. For the sequences under consideration, i.e. those defined by (2.33), the relation $\Delta^2 x_j = -(I - M)\Delta x_j$ holds, and therefore we also have $\Delta^2 X_0 = -(I - M)\Delta X_0$. Furthermore, since we are in effect solving the system $(I - M)x = f$, the residual vector associated with x is $r = f - (I - M)x$. Now consider the residual r_m for the vector y_m:

$$\begin{aligned} r_m &= f - (I - M)y_m = f - (I - M)[x_0 + \Delta X_0 \beta] \\ &= f + Mx_0 - x_0 + \Delta^2 X_0 \beta \\ &= \Delta x_0 + \Delta^2 X_0 \beta. \end{aligned} \tag{2.35}$$

Setting the over-determined system $\Delta x_0 + \Delta^2 X_0 \beta = 0$ yields exactly the same system as (2.24) with $j = 0$.

The idea in the original paper was to determine β so as to minimize the Euclidean norm of the residual (2.35). It is common to formulate this method in the form of a projection technique: the norm $\|\Delta x_0 + \Delta^2 X_0 \beta\|_2$ is minimized by imposing the condition that the residual $r = \Delta x_0 + \Delta^2 X_0 \beta$ is orthogonal to the span of the columns of $\Delta^2 X_0$, i.e. to the second-order forward differences $\Delta^2 x_i$:

$$(\Delta^2 X_0)^\top [\Delta x_0 + \Delta^2 X_0 \beta] = 0. \tag{2.36}$$

This is exactly the same system as in (2.29) with $j = 0$ and $W = \Delta^2 X_0$. Therefore RRE is an instance of the MMPE method seen earlier, where we set W to be $W = \Delta^2 X_0$. The case $m = 1$ is interesting. Indeed, when $m = 1$, equations (2.34) and (2.36) yield the same extrapolation as the vector version of the Aitken process shown in formula (2.21) for the case $j = 0$. One can therefore view the vector version of the Aitken process (2.21) as a particular instance of RRE with a projection dimension m equal to 1.

In the early development of the method, the idea was to exploit a low-rank character of $\Delta^2 X_0$. If $\Delta^2 X_0$ has rank m then y_m will be an exact solution of the system $(I - M)x = f$, since r_m will be zero. Let us explain this in order to make a connection with Krylov subspace methods, to be discussed in Section 3.3. From (2.33) it follows immediately that $x_{j+1} - x_j = M(x_j - x_{j-1})$, and therefore

$$\Delta x_j = x_{j+1} - x_j = M^j (x_1 - x_0) = M^j r_0. \tag{2.37}$$

Therefore the accelerated vector y_m is of the form $y_m = x_0 + q_{m-1}(M)r_0$, where q_{m-1} is a polynomial of degree $\leq m - 1$. The residual vector r_m is $f - (I - M)y_m$ and thus

$$r_m = r_0 - (I - M)q_{m-1}(M)r_0 \equiv p_m(M)r_0, \tag{2.38}$$

where $p_m(t) \equiv 1-(1-t)q_{m-1}(t)$ is a degree m polynomial such that $p_m(1) = 1$. The polynomial q_{m-1}, and therefore also the residual polynomial p_m, is parametrized with the m coefficients $\beta_i, i = 1, \ldots, m$. The minimization problem of RRE can be translated in terms of polynomials: find the degree m polynomial p_m satisfying the constraint $p_m(1) = 1$ for which the norm of the residual $p_m(M)r_0$ is minimal. If the minimal polynomial for M is m then the smallest norm residual is zero. This was behind the motivation of the method. Note that the accelerated solution y_m belongs to the subspace

$$K_m(M, r_0) = \text{Span}\{r_0, Mr_0, \ldots, M^{m-1}r_0\},$$

which is called a *Krylov subspace*. This is the same subspace as the one more commonly associated with linear systems, in which M is replaced by the coefficient matrix $I - M$.

In the scenario where the minimal polynomial is of degree m, the rank of M is also m, hence the term 'reduced rank extrapolation' given to the method. Although the method was initially designed with this special case in mind, it can clearly be used in a more general setting. Note also that although the RRE acceleration scheme was designed for sequences of the form (2.33), the process is an extrapolation method, in that it does not utilize any information other than the original sequence itself. All we need are the first- and second-order difference matrices ΔX_0 and $\Delta^2 X_0$, and the matrix M is never referenced. Furthermore, nothing prevents us from using the procedure to accelerate a sequence produced by some nonlinear fixed-point iteration.

The MPE method mentioned at the beginning of this section is closely related to RRE. Like RRE, MPE expresses the extrapolated solution in the form (2.34) and so the residual of this solution also satisfies (2.35). Instead of trying to minimize the norm of this residual, MPE imposes a Galerkin condition of the form $W^\top(\Delta x_0 + \Delta^2 X_0 \beta) = 0$, but now W is selected to be equal to ΔX_0. As can be seen, this can again be derived from the MMPE framework discussed earlier. Clearly, if M is of rank m then y_m will again be the exact solution of the system. Note also that RRE was presented in difference forms by Eddy (1979) and Mešina (1977), and the equivalence between the two methods was established in Smith, Ford and Sidi (1987).

2.7. *Alternative formulations of RRE and MPE*

In formula (2.34) the accelerated vector y_m is expressed as an update to x_0, the first member of the sequence. It is also possible to express it as an update to x_m, its most recent member. Indeed, the affine space $x_0 + \text{Span}\{\Delta X_0\}$ is identical to $x_m + \text{Span}\{\Delta X_0\}$ because of the relation

$$x_m = x_0 + \Delta x_0 + \Delta x_1 + \cdots + \Delta x_{m-1} = x_0 + \Delta X_0 e,$$

where e is the vector of all ones. If we set $y_m = x_m + \Delta X_0 \gamma$ then the residual in (2.35) becomes $\Delta x_m + \Delta X_0 \gamma$, and so we can reformulate RRE as

$$y_m = x_m + \Delta X_0 \gamma, \quad \text{where } \gamma = \text{argmin}_\gamma \|\Delta x_m + \Delta X_0 \gamma\|_2. \tag{2.39}$$

A similar formulation also holds for MPE. The above formulation will be useful when comparing RRE with the Anderson acceleration to be seen later.

In another expression of MPE and RRE the x_i are invoked directly instead of their differences in (2.34). We will explain this now because similar alternative formulations will appear a few times later in the paper. Equation (2.34) yields

$$\begin{aligned} y_m &= x_0 + \beta_1(x_1 - x_0) + \beta_2(x_2 - x_1) + \cdots + \beta_m(x_m - x_{m-1}) \\ &= (1 - \beta_1)x_0 + (\beta_1 - \beta_2)x_1 + \cdots + (\beta_i - \beta_{i+1})x_i + \cdots + \beta_m x_m. \end{aligned}$$

This can be rewritten as $\sum_{i=0}^m \alpha_i x_i$ by setting $\alpha_i \equiv \beta_i - \beta_{i+1}$ for $i = 0, \ldots, m$, with the convention that $\beta_{m+1} = 0$, and $\beta_0 = 1$. We then observe that the α_i's sum up to one. Thus we can reformulate (2.34) in the form

$$y_m = \sum_{i=0}^m \alpha_i x_i, \quad \text{with } \sum_{i=0}^m \alpha_i = 1. \tag{2.40}$$

The above setting is quite a common alternative to that of (2.34) in acceleration methods. It is also possible to proceed in reverse by formulating an accelerated sequence stated as in (2.40) in the form (2.34), and this was done previously; see Section 2.5.

2.8. Additional comments and references

It should be stressed that extrapolation algorithms of the type discussed in this section often played a major role in providing a framework for the other classes of methods. They were initially invented to deal with scalar sequences, e.g. those produced by quadrature formulas. Later, they served as templates to deal with fixed-point iterations, where the fixed-point mapping g was brought to the fore to develop effective techniques. While 'extrapolation'-type methods were gaining in popularity, physicists and chemists were seeking new ways of accelerating computationally intensive, and slowly converging, fixed-point iterations. The best-known of these fixed-point techniques is the self-consistent field (SCF) iteration, which permeated a large portion of computational chemistry and quantum mechanics. As a background, SCF methods, along with the acceleration tricks developed in the context of the Kohn–Sham equation, will be summarized in Section 4. Solving linear systems of equations is one of the most common practical problems encountered in computational sciences, so it should not be surprising that acceleration methods have also been deployed in this context. The next section addresses the special case of linear systems.

We conclude this section with a few bibliographical pointers. Extrapolation methods generated a rich literature starting in the 1970s, and a number of surveys

and books have appeared that provide a wealth of details, both historical and technical. Among early books on the topic, we mention those of Brezinski (1980) and Brezinski and Redivo-Zaglia (1991). Most of the developments of extrapolation methods took place in the twentieth century, and these are surveyed by Brezinski (2000). A number of other papers provide an in-depth review of extrapolation and acceleration methods; see e.g. Higham and Strabić (2016), Jbilou and Sadok (2000) and Sidi (2012). A nicely written and more recent survey of extrapolation and its relation to rational approximation is the volume by Brezinski and Redivo-Zaglia (2020), which contains a large number of references while also discussing the fascinating lives of the main contributors to these fields.

3. Accelerators for linear iterative methods

Starting with the work of Gauss in 1823 (see Forsythe 1951), quite a few iterative methods for solving linear systems of equations were developed. The idea of accelerating these iterative procedures is natural, and it has been invoked repeatedly in the past. A diverse set of techniques were advocated for this purpose, including Richardson's method, Chebyshev acceleration and the class of Krylov subspace methods. It appears that acceleration techniques were first suggested in the early twentieth century with the work of Richardson, and reappeared in force a few decades later as modern electronic computers started to emerge.

3.1. Richardson's legacy

Consider a linear system of the form

$$Ax = b, \tag{3.1}$$

where $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$. Adopting the point of view expressed in equation (1.3) with the function $f(x) \equiv Ax - b$, we obtain the iteration

$$x_{j+1} = x_j - \gamma(Ax_j - b) = x_j + \gamma r_j, \tag{3.2}$$

where x_j is the current iterate and $r_j = b - Ax_j$ the related residual. This simple 'first-order' scheme was proposed by Richardson (1910). Assuming that the eigenvalues of A are real and included in the interval $[\alpha, \beta]$ with $\alpha > 0$, it is not difficult to see that the scheme will converge to the solution for any γ such that $0<\gamma < 2/\beta$ and that the value of γ that yields the best convergence rate is $\gamma_{opt} = 2/[\alpha+\beta]$; see e.g. Saad (2003).

Richardson also considered a more general procedure where the scalar γ changed at each step:

$$x_{j+1} = x_j + \gamma_j r_j. \tag{3.3}$$

If x_* is the exact solution, he studied the question of selecting the best sequence of γ_j to use if we want the error norm $\|x_j - x_*\|_2$ at the jth step to be the smallest possible. This will be addressed in the next section.

A number of procedures discovered at different times can be cast into the general Richardson iteration framework represented by (3.3). Among these, two examples stand out. The first example is the *minimal residual iteration* (MR), where γ_j is selected as the value of γ that minimizes the next residual norm, $\|b-A(x_j+\gamma r_j)\|_2$. The second is the steepest descent algorithm, where instead of minimizing the residual norm we minimize $\|x - x_*\|_A$, where $\|v\|_A^2 = (Av, v)$. Both methods are one-dimensional projection techniques and will be discussed in Section 3.3.1.

3.2. *The Chebyshev procedure*

Richardson (1910) seems to have been the first to consider a general scheme given by (3.3), where the γ_j are sought with the goal of achieving the fastest possible convergence. From (3.3) we get

$$r_{j+1} = b - A(x_j + \gamma_j r_j) = r_j - \gamma_j A r_j = (I - \gamma_j A) r_j, \tag{3.4}$$

which leads to the relation

$$r_{j+1} = (I - \gamma_j A)(I - \gamma_{j-1} A) \cdots (I - \gamma_0 A) r_0 \equiv p_{j+1}(A) r_0, \tag{3.5}$$

where $p_{j+1}(t)$ is a polynomial of degree $j + 1$. Since $p_{j+1}(0) = 1$, then p_{j+1} is of the form $p_{j+1}(t) = 1 - tq_j(t)$, with deg $(q_j) = j$. Therefore

$$r_{j+1} = (I - Aq_j(A))r_0 = (b - Ax_0) - Aq_j(A)r_0 = b - A[x_0 + q_j(A)r_0],$$

which means that

$$x_{j+1} = x_0 + q_j(A)r_0. \tag{3.6}$$

Richardson worked with error vectors instead of residuals. Defining the error $u_j = x_* - x_j$, where x_* is the exact solution, and using the relation $Au_j = b - Ax_j = r_j$ we can multiply (3.5) by A^{-1} to obtain the relation

$$u_{j+1} = p_{j+1}(A)u_0 \tag{3.7}$$

for the error vector at step $j + 1$, where p_{j+1} is the same polynomial as above. He formulated the problem of selecting the γ_i in (3.7) with a goal of making the error u_{j+1} as small as possible. The γ_i in his formula were the inverses of the ones above, but the reasoning is identical. Such a scheme can be viewed as an 'acceleration' of the first-order Richardson method (3.2) seen earlier. Richardson assumed only the knowledge of an interval $[\alpha, \beta]$ containing the eigenvalues of A. When A is symmetric positive definite (SPD), then there exists α, β, with $\alpha > 0$ such that $\Lambda(A) \subset [\alpha, \beta]$.

If we wish to minimize the maximum deviation from zero in the interval, then the best polynomial can be found from a well-known and simple result in approximation theory. We will reason with residuals, recall equation (3.5), and let $\mathcal{P}_{j+1,0}$ denote the set of polynomials p of degree $j + 1$ such that $p(0) = 1$. Thus p_{j+1} in (3.5) and (3.7) is a member of $\mathcal{P}_{j+1,0}$, and our problem is to find a polynomial $p \in \mathcal{P}_{j+1,0}$

such that $\max_{t\in[\alpha,\beta]} |p(\lambda)|$ is minimal. In other words we seek the solution to the min-max problem

$$\min_{p\in\mathcal{P}_{j+1,0}} \max_{t\in[\alpha,\beta]} |p(t)|. \tag{3.8}$$

The polynomial T_{j+1} that realizes the solution to (3.8) is known, and it can be expressed in terms of the Chebyshev polynomials of the first kind $C_j(t)$:

$$T_{j+1}(t) \equiv \frac{1}{\sigma_{j+1}} C_{j+1}\left(\frac{\beta+\alpha-2t}{\beta-\alpha}\right), \quad \text{with } \sigma_{j+1} \equiv C_{j+1}\left(\frac{\beta+\alpha}{\beta-\alpha}\right). \tag{3.9}$$

If the polynomial T_{j+1} is set to be the same as p_{j+1} in (3.5), then clearly the inverses of its roots will yield the best sequence of γ_i to use in (3.3). Richardson seems to have been unaware of Chebyshev polynomials. Instead, his approach was to select the roots $1/\gamma_i$ by spreading them in an *ad hoc* fashion in $[\alpha, \beta]$. One has to wait more than four decades before this idea, or similar ones based on Chebyshev polynomials, appeared.

A few of the early methods in this context computed the roots of the modified Chebyshev polynomial (3.9) and used the inverses of these roots as the γ_j in (3.3) (Shortley 1953, Sheldon 1955, Young 1954). These methods were difficult to use in practice and prone to numerical instability. A far more elegant approach is to exploit the three-term recurrence of the Chebyshev polynomials:

$$C_{j+1}(t) = 2tC_j(t) - C_{j-1}(t), \quad j \geq 1, \tag{3.10}$$

starting with $C_0(t) = 1, C_1(t) = t$. Lanczos (1952) suggested a process for pre-processing a right-hand side of a linear system prior to solving it with what was then a precursor to a Krylov subspace method. The residual polynomials related to this process are more complicated than Chebyshev polynomials, as was noted by Young (1954). However, they too rely on Chebyshev polynomials, and Lanczos does exploit the three-term recurrence in his 1952 article while Shortley, Sheldon and Young do not.

The first real acceleration scheme based on the Chebyshev polynomials that exploits the three-term recurrence seems to be that of Blair *et al.* (1959). The article included two appendices written by von Neumann, and the second of these discussed the method. The method must have been developed around the year 1956 or earlier by von Neumann, who died on 8 February 1957. Two years after the von Neumann article, Golub and Varga (1961) published a very similar technique which they named the 'semi-iterative method'. They included a footnote acknowledging the earlier contribution by von Neumann.

The arguments of von Neumann's contribution were rooted in acceleration techniques, with a goal of producing a process for linearly combining the previous iterates of a given sequence in order to achieve faster convergence. Specifically, the new sequence is of the form $y_j = \sum_{i=0}^{j} \eta_{i,j} x_i$, where the η satisfy the constraint that $\sum \eta_{i,j} = 1$. Although this may seem different from what was done above, it actually amounts to the same idea.

Indeed, the residual of the 'accelerated' sequence is

$$b - Ay_j = b - \sum_{i=0}^{j} \eta_{i,j} Ax_i = \sum_{i=0}^{j} \eta_{i,j} [b - Ax_i] = \sum_{i=0}^{j} \eta_{i,j} p_i(A) r_0,$$

where $p_i(t)$ is the residual polynomial of degree i associated with the original sequence, that is, it is defined by (3.5). As was already seen, the polynomial p_i satisfies the constraint $p_i(0) = 1$. Because $\sum_{i=0}^{j} \eta_{ij} p_i(0) = \sum \eta_{ij} = 1$, the new residual polynomial $\tilde{p}_j(t) = \sum_{i=0}^{j} \eta_{i,j} p_i(t)$ also satisfies the same condition $\tilde{p}_j(0) = 1$. Therefore we can say that the procedure seeks to find a degree j polynomial, expressed in the form $\sum_{i=0}^{j} \eta_{i,j} p_i(t)$, whose value at zero is one and which is optimal in some sense.

We now return to Chebyshev acceleration to provide details of the procedure discovered by John von Neumann and Golub and Varga. Letting

$$\theta \equiv \frac{\beta + \alpha}{2}, \quad \delta \equiv \frac{\beta - \alpha}{2}, \tag{3.11}$$

we can write T_j defined by (3.9) as

$$T_j(t) \equiv \frac{1}{\sigma_j} C_j \left(\frac{\theta - t}{\delta} \right), \quad \text{with } \sigma_j \equiv C_j \left(\frac{\theta}{\delta} \right). \tag{3.12}$$

The three-term recurrence for the Chebyshev polynomials leads to

$$\sigma_{j+1} = 2 \frac{\theta}{\delta} \sigma_j - \sigma_{j-1}, \quad j = 1, 2 \ldots, \quad \text{with } \sigma_0 = 1, \quad \sigma_1 = \frac{\theta}{\delta}. \tag{3.13}$$

We combine equations (3.10) and (3.12) into a three-term recurrence for the polynomials T_j, for $j \geq 1$:

$$\sigma_{j+1} T_{j+1}(t) = 2 \frac{\theta - t}{\delta} \sigma_j T_j(t) - \sigma_{j-1} T_{j-1}(t), \tag{3.14}$$

starting with $T_0(t) = 1$, $T_1(t) = 1 - t/\theta$.

We now need a way of expressing the sequence of iterates from the above recurrence of the residual polynomials. There are at least two ways of doing this. One idea is to note that $r_{j+1} - r_j = -A(x_{j+1} - x_j) = T_{j+1}(A) r_0 - T_j(A) r_0$. Therefore we need to find a recurrence for $(T_{j+1}(t) - T_j(t))/(-t)$. Going back to (3.14) and exploiting the recurrence (3.13), we write for $j \geq 1$

$$\begin{aligned} \sigma_{j+1} T_{j+1} - \sigma_{j+1} T_j &= 2 \frac{\theta - t}{\delta} \sigma_j T_j - \left(2 \frac{\theta}{\delta} \sigma_j - \sigma_{j-1} \right) T_j - \sigma_{j-1} T_{j-1} \\ &= -2 \frac{t}{\delta} \sigma_j T_j + \sigma_{j-1} (T_j - T_{j-1}) \quad \rightarrow \\ \frac{T_{j+1} - T_j}{-t} &= 2 \frac{1}{\delta} \frac{\sigma_j}{\sigma_{j+1}} T_j + \frac{\sigma_{j-1}}{\sigma_{j+1}} \frac{T_j - T_{j-1}}{-t}. \end{aligned} \tag{3.15}$$

Algorithm 1 Chebyshev acceleration

1: **Input:** System A, b, initial guess x_0 and parameters δ, θ
2: Set $r_0 \equiv b - Ax_0$, $\sigma_1 = \theta/\delta$; $\rho_0 = 1/\sigma_1$ and $d_0 = \frac{1}{\theta} r_0$;
3: **for** $j = 0, 1, \ldots,$ until convergence: **do**
4: $\quad x_{j+1} = x_j + d_j$
5: $\quad r_{j+1} = r_j - Ad_j$
6: $\quad \rho_{j+1} = (2\sigma_1 - \rho_j)^{-1}$
7: $\quad d_{j+1} = \frac{2\rho_{j+1}}{\delta} r_{j+1} + \rho_{j+1}\rho_j d_j$
8: **end for**

When translated into vectors of the iterative scheme, $T_j(t)$ will give r_j, and $(T_{j+1}(t) - T_j(t))/(-t)$ will translate to $x_{j+1} - x_j$, and if we set $d_j = x_{j+1} - x_j$, then (3.15) yields

$$d_j = \frac{2}{\delta}\frac{\sigma_j}{\sigma_{j+1}} r_j + \frac{\sigma_{j-1}}{\sigma_{j+1}} d_{j-1}. \tag{3.16}$$

We can set

$$\rho_j \equiv \frac{\sigma_j}{\sigma_{j+1}}, \quad j = 1, 2, \ldots,$$

and invoke the relation (3.13) to get $\rho_j = 1/[2\sigma_1 - \rho_{j-1}]$, and then (3.16) becomes

$$d_j = \frac{\rho_j}{\delta} r_j + \frac{\rho_j}{\rho_{j-1}} d_{j-1}.$$

This leads to Algorithm 1.

A second way to obtain a recurrence relation for the iterates x_j is to write the error as $x_{j+1} - x_* = T_{j+1}(A)(x_0 - x_*)$ and then exploit (3.14):

$$\sigma_{j+1}(x_{j+1} - x_*) = 2\,\frac{\theta I - A}{\delta}\sigma_j(x_j - x_*) - \sigma_{j-1}(x_{j-1} - x_*) \tag{3.17}$$

$$= 2\frac{\theta}{\delta}\sigma_j(x_j - x_*) + \frac{2\sigma_j}{\delta} r_j - \sigma_{j-1}(x_{j-1} - x_*) \tag{3.18}$$

From the relation (3.13), we see that the terms in x_* cancel out and we get

$$\sigma_{j+1} x_{j+1} = 2\frac{\theta}{\delta}\sigma_j x_j + \frac{2\sigma_j}{\delta} r_j - \sigma_{j-1} x_{j-1}. \tag{3.19}$$

Finally, invoking (3.13) again we can write $2\frac{\theta}{\delta}\sigma_j = \sigma_{j+1} + \sigma_{j-1}$, and hence

$$\sigma_{j+1} x_{j+1} = \sigma_{j+1} x_j + \sigma_{j-1}(x_j - x_{j-1}) + \frac{2\sigma_j}{\delta} r_j. \tag{3.20}$$

Note that lines 4 and 7 of the algorithm can be merged in order to rewrite the iteration as

$$x_{j+1} = x_j + \rho_j\left[\rho_{j-1}(x_j - x_{j-1}) + \frac{2}{\delta}(b - Ax_j)\right], \tag{3.21}$$

which is a 'second-order iteration', of the same class as *momentum-type methods* seen in optimization and machine learning, to be discussed in the next section. This is a common form used in particular by Golub and Varga (1961) in their seminal work on 'semi-iterative methods'. A major advantage of the Chebyshev iterative method is that it does not require any inner products. On the other hand, the scheme requires estimates for the extremal eigenvalue in order to set the sequence of scalars necessary for the iteration.

It is easy to see that the iteration parameters ρ_j used in the algorithm converge to a limit. Indeed, the usual formulas for Chebyshev polynomials show that

$$\sigma_j = \mathrm{ch}\left[j\mathrm{ch}^{-1}\frac{\theta}{\delta}\right] = \frac{1}{2}\left[\left(\frac{\theta}{\delta}+\sqrt{\left(\frac{\theta}{\delta}\right)^2-1}\right)^j + \left(\frac{\theta}{\delta}+\sqrt{\left(\frac{\theta}{\delta}\right)^2-1}\right)^{-j}\right].$$

As a result, we have

$$\lim_{j\to\infty}\rho_j = \lim_{j\to\infty}\frac{\sigma_j}{\sigma_{j+1}} = \left(\frac{\theta}{\delta}+\sqrt{\left(\frac{\theta}{\delta}\right)^2-1}\right)^{-1} = \frac{\theta}{\delta}-\sqrt{\left(\frac{\theta}{\delta}\right)^2-1} \equiv \rho. \quad (3.22)$$

One can therefore consider replacing the scalars ρ_j with their limit in the algorithm. This scheme, which we will refer to as the *stationary Chebyshev iteration*, typically results in a small reduction in convergence speed relative to the standard Chebyshev iteration (Kerkhoven and Saad 1992).

3.3. An overview of Krylov subspace methods

Polynomial iterations of the type introduced by Richardson lead to a residual of the form (3.5), where p_{j+1} is a polynomial of degree $j+1$; see Section 3.1. The related approximate solution at step $j+1$ is given in equation (3.6). The approximate solution x_j at step j is of the form $x_0+\delta$ where δ belongs to the subspace

$$\mathcal{K}_j(A, r_0) = \mathrm{Span}\{r_0, Ar_0, \ldots, A^{j-1}r_0\}. \quad (3.23)$$

This is the *jth Krylov subspace*. As can be seen, $\mathcal{K}_j(A, r_0)$ is simply the space of all vectors of the form $q(A)r_0$, where q is an arbitrary polynomial of degree not exceeding $j-1$. When there is no ambiguity we denote $\mathcal{K}_j(A, r_0)$ by $\mathcal{K}_j$.

Krylov subspace methods for solving a system of the form (3.1) are projection methods on the subspace (3.23) and can be viewed as a form of optimal polynomial acceleration implemented via a projection process. We begin with a brief discussion of projection methods.

3.3.1. Projection methods

The primary objective of a projection method is to extract an approximate solution to a problem from a subspace. Suppose we wish to obtain a solution $x \in \mathbb{R}^n$ to a given problem (P). The problem is projected into a problem $(\tilde{P})$ set in a subspace $\mathcal{K}$ of $\mathbb{R}^n$ from which we obtain an approximate solution $\tilde{x}$. Typically, the dimension m of $\mathcal{K}$ is much smaller than n, the size of the system.

When applied to the solution of linear systems of equations, we assume the knowledge of some initial guess x_0 to the solution and two subspaces $\mathcal{K}$ and $\mathcal{L}$, both of dimension m. From these the following projected problem is formulated:

$$\text{Find } \tilde{x} = x_0 + \delta,\ \delta \in \mathcal{K} \text{ such that } b - A\tilde{x} \perp \mathcal{L}. \tag{3.24}$$

We have m degrees of freedom (dimension of $\mathcal{K}$) and m constraints (dimension of $\mathcal{L}$), and so (3.24) will result in an $m \times m$ linear system which is non-singular under certain mild conditions on $\mathcal{K}$ and $\mathcal{L}$. The Galerkin projection process just described satisfies important *optimality properties* that play an essential role in their analysis. There are two important cases.

Orthogonal projection (OP) methods. This case corresponds to the situation where the subspaces $\mathcal{K}$ and $\mathcal{L}$ are the same. In this scenario the residual $b-A\tilde{x}$ is orthogonal to $\mathcal{K}$ and the corresponding approximate solution is the closest vector from affine space $x_0 + \mathcal{K}$ to the exact solution x_*, where distance is measured with the A-norm, $\|v\|_A = (Av, v)^{1/2}$.

Proposition 3.1. Assume that A is symmetric positive definite and $\mathcal{L} = \mathcal{K}$. Then a vector $\tilde{x}$ is the result of an (orthogonal) projection method onto $\mathcal{K}$ with the starting vector x_0 if and only if it minimizes the A-norm of the error over $x_0 + \mathcal{K}$, i.e. if and only if

$$E(\tilde{x}) = \min_{x \in x_0 + \mathcal{K}} E(x),$$

where

$$E(x) \equiv (A(x_* - x), x_* - x)^{1/2}.$$

The necessary condition means the following:

$$\mathcal{L} = \mathcal{K} \quad \text{and} \quad A \text{ SPD} \longrightarrow \|x_* - \tilde{x}\|_A = \min_{x \in x_0 + \mathcal{K}} \|x_* - x\|_A.$$

It is interesting to note that Richardson's scheme shown in equation (3.3) can be cast to include another prominent one-dimensional projection method, namely the well-known steepest descent algorithm. Here, A is assumed to be symmetric positive definite, and at each step the algorithm computes the vector of the form $x = x_j + \gamma r_j$, where $r_j = b - Ax_j$, that satisfies the orthogonality condition $b - Ax \perp r_j$. This leads to selecting at each step $\gamma = \gamma_j \equiv (r_j, r_j)/(Ar_j, r_j)$, which, according to the above proposition, minimizes $\|x - x_*\|_A$ over γ. Another method in this category is the conjugate gradient method, which will be covered shortly.

Minimal residual (MR) methods. This case corresponds to the situation when $\mathcal{L} = A\mathcal{K}$. It can be shown that if A is non-singular, then $\tilde{x}$ minimizes the Euclidean norm of the residual over the affine space $x_0 + \mathcal{K}$.

Proposition 3.2. Let A be an arbitrary square matrix and assume that $\mathcal{L} = A\mathcal{K}$. Then a vector $\tilde{x}$ is the result of an (oblique) projection method onto $\mathcal{K}$ orthogonally

Algorithm 2 Conjugate gradient

1: Compute $r_0 := b - Ax_0$, $p_0 := r_0$.
2: **for** $j = 0, 1, \ldots,$ until convergence Do: **do**
3: $\alpha_j := (r_j, r_j)/(Ap_j, p_j)$
4: $x_{j+1} := x_j + \alpha_j p_j$
5: $r_{j+1} := r_j - \alpha_j A p_j$
6: $\beta_j := (r_{j+1}, r_{j+1})/(r_j, r_j)$
7: $p_{j+1} := r_{j+1} + \beta_j p_j$
8: **end for**

to $\mathcal{L}$ with the starting vector x_0 if and only if it minimizes the 2-norm of the residual vector $b - Ax$ over $x \in x_0 + \mathcal{K}$, i.e. if and only if

$$R(\tilde{x}) = \min_{x \in x_0 + \mathcal{K}} R(x),$$

where $R(x) \equiv \|b - Ax\|_2$.

The necessary condition now means the following:

$$\mathcal{L} = A\mathcal{K} \quad \text{and} \quad A \text{ non-singular} \longrightarrow \|b - A\tilde{x}\|_2 = \min_{x \in x_0 + \mathcal{K}} \|b - Ax\|_2.$$

Methods in this category include the conjugate residual (CR) method, the generalized conjugate residual (GCR) method, and GMRES among others; see Section 3.3.3.

Another instance of Richardson's general iteration of the form (3.3) is the *minimal residual iteration* (MR), where γ_j is selected as the value of γ that minimizes the next residual norm, $\|b - A(x_j + \gamma r_j)\|_2$. A little calculation will show that the optimal γ is $\gamma_j = (r_j, Ar_j)/(Ar_j, Ar_j)$, where it is assumed that $Ar_j \neq 0$; see e.g. Saad (2003). MR is a one-dimensional projection method since it computes a vector of the form $x_j + d$, where $d \in \mathcal{X} = \text{Span}\{r_j\}$, which satisfies the orthogonality condition $b - Ax \perp A\mathcal{X}$.

3.3.2. OP-Krylov and the conjugate gradient method

One particularly important instance in the OP class of projection methods is the *conjugate gradient algorithm* (CG) algorithm, a clever implementation of the case where $\mathcal{K} = \mathcal{L}$ is a Krylov subspace of the form (3.23) and A is SPD. This implementation is shown in Algorithm 2.

The discovery of CG (Hestenes and Stiefel 1952) was a major breakthrough in the early 1950s. Magnus Hestenes (UCLA) and Eduard Stiefel (ETH) made the discovery independently and, as they learned of each other's work, decided to publish the paper together; see Saad (2022) for a brief history of Krylov methods.

Today the CG algorithm is often presented as a projection method on Krylov subspaces, but Hestenes and Stiefel (1952) invoked purely geometric arguments, as their insight from the two-dimensional case and knowledge of 'conics' led

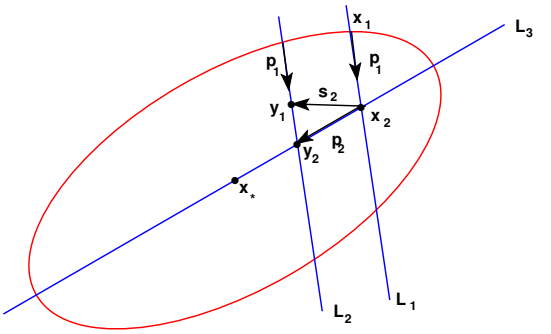


Figure 3.1. Illustration of the construction of conjugate gradient directions. This illustration is based on Hestenes and Todd (1991).

them to the notion of *conjugate directions*. The goal is to find the minimum of $f(x) = \frac{1}{2}(Ax, x) - (b, x)$. In $\mathbb{R}^2$ the contour lines of $f(x)$, i.e. the sets $\{x | f(x) = \kappa\}$ where κ is a constant, are con-focal ellipses, the centre of which is x_*, the desired solution.

Figure 3.1 provides an illustration borrowed from Hestenes and Todd (1991). The minimum of $f(x)$ on a given chord of the ellipse is reached in the middle of the chord. The middles x_2, y_2 of two given parallel chords L_1, L_2 will define a line L_3 that passes through the centre x_* of the ellipse. The minimum of $f(x)$ along L_3 will be at the centre, i.e. at the exact solution. If $x_2 = x_1 + \alpha_1 p_1$ is an iterate and $y_1 = x_2 + s_2$, an intermediate iterate, the line L_2, $y(t) = y_1 + t p_1$, is parallel to L_1. Its minimum is reached at a point $y_2 = y_1 + \beta_1 p_1$, and the direction $p_2 = y_2 - x_2 \equiv p_1 + \beta_1 s_2$ is conjugate to p_1. In the CG algorithm, the direction s_2 is taken to be the residual r_2 of x_2.

At the same time as the CG work was unfolding, Cornelius Lanczos, who also worked at the INA, developed a similar method using a different notation and viewpoint (Lanczos 1952). This was a minimal residual approach (MR method, discussed below), implemented with the use of what we now call a Lanczos basis (Lanczos 1950), for solving eigenvalue problems – another major contribution to numerical linear algebra from the same era.

The conjugate gradient algorithm was not too well received initially. It was viewed as an unstable direct solution method, and for this reason it lay dormant for about two decades. Two important articles played a role in its revival. The first one was by John Reid (1971), who discovered that the method could be rather effective when used as an iterative procedure. At the same time, work by Paige (1971, 1980) analysed the Lanczos process for eigenvalue problems from the point of view of inexact arithmetic, and this work gave a much better understanding of how the loss of orthogonality affected the process. The same loss of orthogonality affected the CG method and led to its initial rejection by numerical analysts.

3.3.3. MR-Krylov methods, GCR, GMRES

Quite a few projection methods on the Krylov subspaces $\mathcal{K}_j(A, r_0)$ were developed starting in the late 1970s, with the objective of minimizing the residual norm. As was seen earlier, this optimality condition is equivalent to the property that the approximate solution $\tilde{x} \in \mathcal{K}_j(A, r_0)$ is orthogonal to the subspace $A\mathcal{K}_j(A, r_0)$.

Implementations with orthonormal basis of K_m led to the generalized minimal residual (GMRES) method (Saad and Schultz 1986), a procedure based on the Arnoldi process. Other implementations included Axelsson's GCG-LS method (Axelsson 1980), ORTHOMIN (Vinsome 1976), ORTHODIR (Jea and Young 1980) and the generalized conjugate residual method (GCR) of Eisenstat, Elman and Schultz (1983), among others. A fairly exhaustive treatise on this work can be found in the book by Meurant and Tebbens (2020). We now briefly discuss the GCR method for solving the system (3.1), since this approach will be exploited later.

The original GCR algorithm (Eisenstat *et al.* 1983) for solving (3.1) exploits an orthogonal basis of the subspace $A\mathcal{K}_j(A, r_0)$. The jth step of the procedure can be described as follows. Assume that we already have vectors $p_0, p_0, \ldots, p_j$ that are $A^\top A$-orthogonal, i.e. such that $(Ap_i, Ap_k) = 0$ if $i \neq k$ for $i, k \leq j$. Thus the set $\{p_i\}_{i=0,\ldots,j}$ forms an $A^\top A$ -orthogonal basis of the Krylov subspace $\mathcal{K}_{j+1}$. In this situation the solution x_{j+1} at the current iteration, i.e. the one that minimizes the residual norm in $x_0 + \mathcal{K}_{j+1} = x_0 + \mathrm{Span}\{p_0, p_1, \ldots, p_j\}$, becomes easy to express. It can be written as follows (see Lemma 6.21 of Saad 2003, and Lemma 3.1 below):

$$x_{j+1} = x_j + \frac{(r_j, Ap_j)}{(Ap_j, Ap_j)} p_j, \tag{3.25}$$

where x_j is the current iterate and r_j is the related residual $r_j = b - Ax_j$.

Once x_{j+1} is computed, the next basis vector, i.e. p_{j+1}, is obtained from $A^\top A$-orthogonalizing the residual vector $r_{j+1} = b - Ax_{j+1}$ against the previous p_i by the loop

$$p_{j+1} = r_{j+1} - \sum_{i=0}^{j} \beta_{ij} p_i, \quad \text{where } \beta_{ij} := (Ar_{j+1}, Ap_i)/(Ap_i, Ap_i). \tag{3.26}$$

The above succinctly describes one step of the algorithm. In practice, it is necessary to keep a set of vectors for the p_i's and another set for the Ap_i's. We will set $v_i = Ap_i$ in what follows. In the classical ('full') version of GCR, these sets are denoted by

$$P_j = [p_0, p_1, \ldots, p_j], \quad V_j = [v_0, v_1, \ldots, v_j]. \tag{3.27}$$

Note here that *all* previous search directions p_i and the corresponding v_i must be saved in this full version.

We will bring two modifications to the basic procedure just described. The first is to introduce a *truncated* version of the algorithm whereby only the most recent $\min\{m, j+1\}$ vectors $\{p_i\}$ and $\{Ap_i\}$ are kept. Thus the summation in (3.26)

Algorithm 3 TGCR(m)

1: **Input:** Matrix A, right-hand side b, initial guess x_0, window size m
2: Set $r_0 \equiv b - Ax_0$; $v = Ar_0$;
3: $v_0 = v/\|v\|_2$; $p_0 = r_0/\|v\|_2$;
4: **for** $j = 0, 1, 2, \ldots,$ Until convergence **do**
5: $\alpha_j = \langle r_j, v_j \rangle$
6: $x_{j+1} = x_j + \alpha_j p_j$
7: $r_{j+1} = r_j - \alpha_j v_j$
8: $p = r_{j+1}$; $v = Ap$;
9: **if** j> 0 **then** [with V_j, P_j defined in (3.28)]
10: Compute $s = V_j^\top v$
11: Compute $v = v - V_j s$ and $p = p - P_j s$
12: **end if**
13: $p_{j+1} := p/\|v\|_2$; $v_{j+1} := v/\|v\|_2$;
14: **end for**

starts at $i = [j - m + 1]$ instead of $i = 0$, that is, it starts at $i = 0$ for $j < m$ and at $i = j - m + 1$ otherwise. The sets P_j, V_j in (3.27) are replaced by

$$P_j = [p_{[j-m+1]}, \ldots, p_j], \quad V_j = [v_{[j-m+1]}, \ldots, v_j]. \tag{3.28}$$

The second change is to make the set of v_i *orthonormal*, i.e. such that $(v_j, v_i) = \delta_{ij}$. Thus the new vector v_{j+1} is made orthonormal to $v_{j-m+1}, v_{j-m+2}, \ldots, v_j$, when $j - m + 1 \geq 0$ or to $v_0, v_1, \ldots, v_j$ otherwise.

Regarding notation, it may be helpful to observe that the last column of P_j (resp. V_j) is always p_j (resp. v_j) and that the number of columns of P_j (and also V_j) is $\min\{m, j + 1\}$. Note also that in practice it is preferable to replace the orthogonalization steps in lines 10–11 of the algorithm with a modified Gram–Schmidt procedure.

The following lemma explains why the update to the solution takes the simple form of equation (3.25). This result was stated in a slightly different form in Saad (2003, Lemma 6.21).

Lemma 3.1. Assume that we are solving the linear system $Ax = b$ and let $\{p_i, v_i\}_{i=[j-m+1]:j}$ be a paired set of vectors with $v_i = Ap_i, i = [j - m + 1], \ldots, j$. Assume also that the set V_j is orthonormal. Then the solution vector of the affine space $x_j + \text{Span}\{P_j\}$ with smallest residual norm is $x_j + P_j y_j$, where $y_j = V_j^\top r_j$. In addition, only the bottom component of y_j, namely $v_j^\top r_j$, is non-zero.

Proof. The residual of $x = x_j + P_j y$ is $r = r_j - AP_j y = r_j - V_j y$ and its norm is smallest when $V_j^\top r = 0$. Hence the first result. Next, this condition implies that the inner products $v_i^\top r_{j+1}$ are all zero when $i \leq j$, so the vector $y_{j+1} = V_{j+1}^\top r_{j+1}$ at the next iteration will have only one non-zero component, namely its last one, i.e. $v_{j+1}^\top r_{j+1}$. This proves the second result for the index $j + 1$ replaced by j. □

It follows from the lemma that the approximate solution obtained by a projection method that minimizes the residual norm in the affine space $x_j + \text{Span}\{P_j\}$ can be written in the simple form $x_{j+1} = x_j + \alpha_j p_j$, where $\alpha_j = v_j^\top r_j$. This explains formula (3.25) shown above and lines 5–6 of Algorithm 3 when we take into consideration the orthonormality of the v_i.

When there is no truncation, we call this the 'full window' case of the algorithm. This is equivalent to setting $m = \infty$ in the algorithm. The truncated variation to GCR ($m < \infty$), which we will call truncated GCR (TGCR), was first introduced in Vinsome (1976) and was named 'ORTHOMIN'. Eisenstat *et al.* (1983) established a number of results for both the full window case ($m = \infty$) and the truncated case ($m < \infty$), including a convergence analysis for the situation when A is positive real, i.e. when its symmetric part is positive definite. In addition to the orthogonality of the vectors Ap_i, or equivalently the $A^\top A$-orthogonality of the p_i, another property of (full) GCR is that the residual vectors that it produces are 'semi-conjugate' in that $(r_i, Ar_j) = 0$ for $i > j$.

Note that when $j \geq m$ in TGCR ($m < \infty$), then the approximate solution x_j no longer minimizes the residual norm over the whole Krylov subspace $x_0 + \mathcal{K}_j(A, r_0)$ but only over $x_{j-m} + \text{span}\{p_{j-m}, \ldots p_j\}$; see Eisenstat *et al.* (1983, Theorem 4.1).

3.4. *Momentum-based techniques*

The Chebyshev iteration provides a good introduction to the notion of *momentum*. It is sufficient to frame the method for an optimization problem, where we seek to minimize the quadratic function $\phi(x) = \frac{1}{2}x^\top Ax - b^\top x$, where A is SPD. In this case $\nabla\phi(x) = Ax - b$, which is the negative of the residual. With this we see that Chebyshev iteration (3.21) can be written as

$$x_{j+1} = x_j + \eta_j \Delta x_{j-1} - \nu_j \nabla\phi(x_j), \tag{3.29}$$

where $\eta_j = \rho_j \rho_{j-1}$ and $\nu_j = 2\rho_j/\delta$. Recall the notation $\Delta x_{j-1} = x_j - x_{j-1}$.

3.4.1. *The 'heavy ball' method*

Equation (3.29) is the general form of a gradient-type method *with momentum*, whereby the next iterate x_{j+1} is a combination of the term $x_j - \nu_j \nabla\phi(x_j)$, which can be viewed as a standard gradient-type iterate, and the previous increment, i.e. the difference $x_j - x_{j-1}$. This difference Δx_{j-1} is often termed *velocity* and denoted by v_{j-1} in the literature. Thus a method with momentum takes the gradient iterate from the current point and adds a multiple of *velocity*. A comparison is often made with a mechanical system representing a ball rolling downhill. Often the term *heavy ball* method is used to describe the iteration (3.29) in which the coefficients η_j, μ_j are constant. An illustration is provided in Figure 3.2.

It is common to rewrite equation (3.29) by explicitly invoking a momentum part. This can be done by defining $v_j \equiv \Delta x_j = x_{j+1} - x_j$. Then the update (3.29) can be

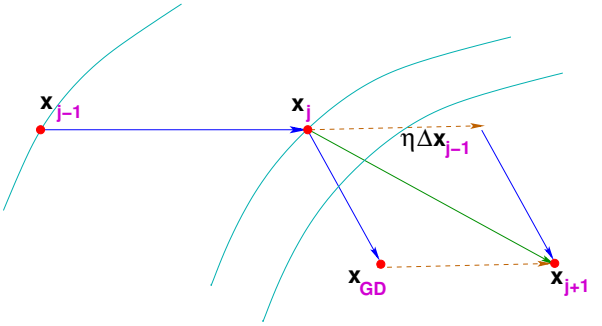


Figure 3.2. Illustration of the gradient method with momentum.

written in two parts as

$$\begin{cases} v_j &= \eta_j v_{j-1} - \nu_j \nabla\phi(x_j), \\ x_{j+1} &= x_j + v_j. \end{cases} \tag{3.30}$$

However, often the velocity v_j is defined with the opposite sign,[2] i.e. $v_j \equiv -\Delta x_j = x_j - x_{j+1}$, in which case the update (3.29) becomes

$$\begin{cases} v_j &= \eta_j v_{j-1} + \nu_j \nabla\phi(x_j), \\ x_{j+1} &= x_j - v_j. \end{cases} \tag{3.31}$$

Both expressions (3.30) and (3.31) can be found in the literature but we will utilize (3.31), which is equivalent to a form seen in machine learning (ML). In ML, the scalar parameters of the sequence are constant, i.e. $\eta_j \equiv \eta$, $\nu_j \equiv \nu$, and the velocity vector v_j is often scaled by ν, that is, we set $v_j = \nu w_j$, upon which (3.31) becomes

$$\begin{cases} w_j &= \eta w_{j-1} + \nabla\phi(x_j), \\ x_{j+1} &= x_j - \nu w_j. \end{cases} \tag{3.32}$$

In this way, w_j is just a damped average of previous gradients, where the damping coefficient is a power of η that gives more weight to recent gradients. In deep learning, the gradient is actually a sampled gradient corresponding to a 'batch' of functions that make up the final objective function; see Section 8.

3.4.2. Convergence

The convergence of Chebyshev iteration for linear problems is well understood; see e.g. Saad (2011). Here we consider, more generally, the momentum scheme (3.29), but we restrict our study to the particular case where the scalars η_j, ν_j are constant, denoted by η, ν respectively. We also make the same assumption as for

[2] The motivation here is that when $\eta = 0$, which corresponds to the gradient method without momentum, the vector v_j in the update $x_{j+1} = x_j + v_j$ should be a negative multiple of the gradient of ϕ, so changing the sign makes v_j a positive multiple of the gradient.

the Chebyshev iteration that the eigenvalues of A are real and located in the interval $[\alpha, \beta]$ with $\alpha > 0$. Equation (3.29) becomes

$$\begin{aligned} x_{j+1} &= x_j + \eta(x_j - x_{j-1}) - \nu(Ax_j - b) \\ &= [(1+\eta)I - \nu A]x_j - \eta x_{j-1} + \nu b. \end{aligned} \tag{3.33}$$

To analyse the convergence of the above iteration, we can write it in the form

$$\begin{pmatrix} x_{j+1} \\ x_j \end{pmatrix} = \left(\begin{array}{c|c} (1+\eta)I - \nu A & -\eta I \\ \hline I & 0 \end{array}\right) \begin{pmatrix} x_j \\ x_{j-1} \end{pmatrix} + \begin{pmatrix} \nu b \\ 0 \end{pmatrix}. \tag{3.34}$$

It is helpful to introduce two matrices,

$$B = \frac{1}{\delta}(A - \theta I) \quad \text{and} \quad C = \frac{1}{2}[(1+\eta)I - \nu A], \tag{3.35}$$

where we recall that δ, θ are defined in (3.11) and that the eigenvalues $\lambda_i(B)$ are in the interval $[-1, 1]$. Then the iteration matrix in (3.34) is

$$G = \begin{pmatrix} 2C & -\eta I \\ I & 0 \end{pmatrix}. \tag{3.36}$$

If μ_j, $j = 1, \ldots, n$, are the eigenvalues of C, then those of G are

$$\lambda_j = \mu_j \pm \sqrt{\mu_j^2 - \eta}. \tag{3.37}$$

This expression can help determine if the scheme (3.29) will converge. As a particular case, a sufficient condition for convergence is that $0 < \eta < 1$ and all μ_j are less than $\sqrt{\eta}$ in magnitude. In this case $\mu_j^2 - \eta$ is negative and the modulus of λ_j is a constant equal to $\sqrt{\eta}$, which is independent of j.

As an example, we will look at what happens in the case of the stationary Chebyshev iteration defined earlier.

Proposition 3.3. Consider the stationary Chebyshev iteration in which $\eta_j = \eta \equiv \rho^2$ and $\nu_j = \nu = 2(\rho/\delta)$, where ρ was defined in (3.22). Then each of the eigenvalues μ_j of the matrix C in (3.35) satisfies the inequality $|\mu_j| \leq \rho$. In addition, if $\rho < 1$ the stationary Chebyshev iteration converges and its convergence factor, that is, the spectral radius of the matrix G in (3.36), is equal to ρ.

Proof. The eigenvalues μ_j of C are related to those of B as follows:

$$\mu_j = \frac{1}{2}[1 + \eta - \nu\lambda_j(A)] = \frac{1}{2}[1 + \eta - \nu(\theta + \delta\lambda_j(B))]. \tag{3.38}$$

Substituting $\eta = \rho^2$ and $\nu = 2\rho/\delta$ leads to

$$\mu_j = \frac{1}{2}\left[1 + \rho^2 - \frac{2\rho}{\delta}(\theta + \delta\lambda_j(B))\right] = \frac{1}{2}\left[\rho^2 - 2\frac{\theta}{\delta}\rho + 1\right] - \rho\lambda_j(B). \tag{3.39}$$

It can be observed that ρ in (3.22) is a root of the quadratic term in the brackets in the second part, i.e. $\rho^2 - 2\frac{\theta}{\delta}\rho + 1 = 0$, and so $\mu_j = -\rho\lambda_j(B)$. Since $|\lambda_j(B)| \leq 1$, it is clear that $|\mu_j| \leq \rho$ and therefore the term $\mu_j^2 - \eta = \mu_j^2 - \rho^2$ in (3.37) is non-positive. Hence the eigenvalues λ_j are of the form $\lambda_j = \mu_j \pm i\sqrt{\rho^2 - \mu_j^2}$ and they all have the same modulus ρ. □

It may seem counter-intuitive that a simple fixed-point iteration like (3.34) can be competitive with more advanced schemes, but we note that we have doubled the dimension of the problem relative to a simple first-order scheme. The strategy of moving a problem into a higher dimension to achieve better convergence is common.

3.4.3. Nesterov acceleration

A slight variation of the momentum scheme discussed above is *Nesterov*'s iteration (Nesterov 2014). In this approach the gradient is evaluated at an intermediate point instead of the most recent iterate:

$$x_{j+1} = x_j + \eta_j \Delta x_{j-1} - \nu_j \nabla\phi(x_j + \eta_j \Delta x_{j-1}). \tag{3.40}$$

Using earlier notation where $v_{j-1} = -\Delta x_{j-1}$, this can also be rewritten as

$$\begin{cases} v_j &= \eta_j v_{j-1} + \nu_j \nabla\phi(x_j - \eta_j v_{j-1}), \\ x_{j+1} &= x_j - v_j. \end{cases} \tag{3.41}$$

To analyse convergence in the linear case, we need to rewrite (3.40) in a block form similar to (3.34). Setting $\eta_j = \eta$, $\nu_j = \nu$ and $\nabla\phi(x) = Ax - b$ in (3.40) yields

$$\begin{aligned} x_{j+1} &= x_j + \eta(x_j - x_{j-1}) - \nu[A(x_j + \eta(x_j - x_{j-1})) - b] \\ &= (1+\eta)x_j - \eta x_{j-1} - \nu(1+\eta)Ax_j + \nu\eta Ax_{j-1} + \nu b \\ &= (1+\eta)(I - \nu A)x_j - \eta[I - \nu A]x_{j-1} + \nu b, \end{aligned}$$

which leads to

$$\begin{pmatrix} x_{j+1} \\ x_j \end{pmatrix} = \left(\begin{array}{c|c} (1+\eta)(I - \nu A) & -\eta(I - \nu A) \\ \hline I & 0 \end{array}\right) \begin{pmatrix} x_j \\ x_{j-1} \end{pmatrix} + \begin{pmatrix} \nu b \\ 0 \end{pmatrix}. \tag{3.42}$$

The scheme represented by (3.40) and its matrix form (3.42) can be viewed as a way of accelerating a Richardson-type scheme (see (3.2)), with the parameters γ_j set equal to ν for all j. The corresponding iteration matrix (3.4) is the matrix $B \equiv I - \nu A$. If $\mu_j, j = 1, \ldots, n$ are the eigenvalues of B, those of the iteration matrix in (3.42) are roots of the equation $\lambda^2 - (1+\eta)\mu_j\lambda + \eta\mu_j = 0$, and they can be written as

$$\lambda_j^{\pm} = \mu_j \left[\frac{1+\eta}{2} \pm \sqrt{\left(\frac{1+\eta}{2}\right)^2 - \frac{\eta}{\mu_j}}\right], \tag{3.43}$$

with the convention that when $\mu_j = 0$ both roots are zero. Assume that the eigenvalues of B are real and such that

$$-\theta_1 \leq \mu_j \leq \theta_2, \tag{3.44}$$

where θ_1, θ_2 are non-negative. It is convenient to define

$$\theta_* = \frac{\eta}{\frac{1}{2}(1+\eta)^2}. \tag{3.45}$$

The simplest scenario to analyse is when ν and η are selected such that $\theta_1 = 0$ and $\theta_2 = \theta_*$. For example, we can first set $\nu = 1/\lambda_{\max}(A)$ to satisfy the requirement $\theta_1 = 0$ since in this case

$$\mu_i = 1 - \nu\lambda_i(A) \geq 1 - \frac{\lambda_{\max}}{\lambda_{\max}} = 0.$$

Then, with ν set to the value just selected, we will find η so that $\theta_* = \theta_2$, i.e. $\eta/(\frac{1}{2}(1+\eta))^2 = \theta_2$, which yields the quadratic equation

$$\eta^2 - 2\left(\frac{2}{\theta_2} - 1\right)\eta + 1 = 0. \tag{3.46}$$

Note that if the eigenvalues of A are positive, then $\theta_2 \leq 1$ and the discriminant $\Delta = (2/\theta_2 - 1)^2 - 1$ is non-negative, so the roots are real. It is important to also note that the product of the two roots of this equation is one, and we will select η to be the smallest of the two roots, so we know it will not exceed one.

In this scenario, the eigenvalues μ_j are in the interval $[0,\ \theta_*]$ and will yield a negative value inside the square root in formula (3.43) for $\lambda_j^{\pm}$. The squared modulus of $\lambda_j^{\pm}$ is

$$\mu_j^2\left[\left(\frac{1+\eta}{2}\right)^2 + \frac{\eta}{\mu_j} - \left(\frac{1+\eta}{2}\right)^2\right] = \eta\mu_j. \tag{3.47}$$

So each of these eigenvalues is transformed into a complex conjugate pair of eigenvalues with modulus $\sqrt{\eta\mu_j}$, but since $\mu_j \leq \theta_*$ the maximum modulus is $\leq 2\eta/(1+\eta)$, which is less than one when μ is selected to be the smallest root of (3.46) as discussed above. In this situation the scheme will converge with a convergence factor $2\eta/(1+\eta)$.

It is also possible to analyse convergence in a more general scenario when

$$-\theta_1 \leq 0 \leq \theta_* \leq \theta_2.$$

In this situation we would need to distinguish between three different cases corresponding to the intervals $[\theta_1,\ 0)$, $[0,\ \theta_*)$ and $[\theta_*,\ \theta_2]$. The analysis is more complex and is omitted.

4. Accelerating the SCF iteration

The Schrödinger equation $H\Psi = E\Psi$ was the result of a few decades of exceptionally productive research starting in the late nineteenth century that revolutionized our understanding of matter at the nanoscale; see Gamov (1966). In this equation E is the energy of the system and $\Psi = \Psi(r_1, r_2, \ldots, r_n, R_1, R_2, \ldots, R_N)$ is the wavefunction that depends on the positions R_i, r_i of the nuclei and electrons respectively. This dependence makes the equation intractable except for the smallest cases. Thus Dirac (1929, p. 1) famously stated:

> The underlying physical laws necessary for the mathematical theory of a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that the exact application of these laws leads to equations much too complicated to be soluble. It therefore becomes desirable that approximate practical methods of applying quantum mechanics should be developed.

It took about four additional decades to develop such methods, a good representative of which is *density functional theory* (DFT).

4.1. The Kohn–Sham equations and the SCF iteration

DFT manages to obtain a tractable version of the original equation by replacing the many-particle wavefunction Ψ with one that depends on one fictitious particle which will generate the same charge density as that of the original interacting multi-particle system. The foundation of the method is a fundamental result by Kohn and Sham (1965), namely that observable quantities are uniquely determined by the ground state charge density. The resulting Kohn–Sham equation can be written as

$$\left[-\frac{h^2}{2m}\nabla^2 + V_{tot}[\rho(r)]\right]\Psi(r) = E\Psi(r), \tag{4.1}$$

where the total potential V_{tot} is the sum of three terms:

$$V_{tot} = V_{ion} + V_H + V_{xc}. \tag{4.2}$$

Both the Hartree potential V_H and the exchange-correlation V_{xc} depend on the charge density (or electron density), which is defined from 'occupied' wavefunctions:

$$\rho(r) = \sum_{i}^{occup} |\Psi_i(r)|^2. \tag{4.3}$$

Thus the Hartree potential V_H is a solution of the Poisson equation, $\nabla^2 V_H = -4\pi\rho(r)$, on the domain of interest, while the exchange-correlation potential depends nonlinearly on ρ under various expressions that depend on the model. Once discretized, equation (4.1) yields a large eigenvalue problem, typically involving n_{occup} eigenvalues and associated eigenvectors. For additional details, see Saad, Chelikowsky and Shontz (2009).

The basic SCF iteration would start from a guessed value of the charge density and other variables and obtain a total potential from it. Then the eigenvalue problem is solved to give a new set of wavefunctions ψ_i, from which a new charge density ρ is computed, and the cycle is repeated until self-consistency is reached, i.e. until the input ρ_{in} and output ρ_{out} are close enough. One can capture the process of computing ρ in this way by a fixed-point iteration,

$$\rho_{j+1} = g(\rho_j), \tag{4.4}$$

where g is a rather complex mapping that computes a new charge density from the old one. This mapping involves solving an eigenvalue problem, a Poisson equation, and making some other updates.

Convergence of this process can be slow in some specific situations. In addition, the cost of each iteration can be enormous. For example, one could have a discretized eigenvalue problem with a size in the tens of millions, and a number of occupied states, i.e. eigenpairs to compute, in the tens of thousands. It is therefore natural to think of employing procedures that accelerate the process, and this can be done in a number of ways.

4.2. Simple mixing

The idea of acceleration used in the context of DFT is different from that of extrapolation discussed earlier. Here researchers invoke the idea of 'mixing' a current ρ with older ones. The most basic of these is known as 'simple mixing' and it defines a new ρ from an old one as follows, where γ is a scalar parameter:

$$\rho_{j+1} = \gamma g(\rho_j) + (1-\gamma)\rho_j \tag{4.5}$$

$$= \rho_j + \gamma(g(\rho_j) - \rho_j). \tag{4.6}$$

It may be helpful to link the above acceleration scheme with other methods that have been developed in different contexts, and to this end a suitable notation is key to unravelling these links. We will replace ρ with a more general variable x, so (4.4) becomes

$$x_{j+1} = g(x_j), \tag{4.7}$$

and we will let f denote the function

$$f(x) = g(x) - x. \tag{4.8}$$

With this, the 'simple mixing' iteration (4.6) becomes

$$x_{j+1} = x_j + \gamma f(x_j). \tag{4.9}$$

The first link that will help set the notation and terminology is with Richardson's iteration (see equation (3.2)) for solving linear systems of the form $Ax = b$. In this case $g(x) = x + \gamma(b - Ax)$ and $f(x) = b - Ax$. So Richardson's iteration amounts to the simple mixing scheme above, where $f(x)$ *is the residual* $b - Ax$. We will refer to the general scheme defined by (4.9) as Richardson's iteration, or 'linear mixing',

in order to avoid the term 'simple mixing', which is proper to the physics literature. Often, linear iterations are expressed in the form $x_{j+1} = Mx_j + b$. In this case we seek to solve the system $(I - M)x = b$, and $f(x) = (I - M)x - b$ is the negative of the residual.

The next link is with gradient descent algorithms for minimizing a scalar function $\phi(x)$. Assuming that the function is convex and differentiable, the iteration reads

$$x_{j+1} = x_j - \gamma \nabla \phi(x_j), \tag{4.10}$$

where γ is a positive scalar selected to ensure a decrease of the function $\phi(x)$. In the special case where $\phi(x) = \frac{1}{2}x^\top Ax - b^\top x$ and A is symmetric, then $\nabla\phi(x) = Ax - b$, which is the negative of the residual, i.e. $\nabla\phi(x) = -(b - Ax)$. This leads to the gradient method for minimizing $\phi(x)$, and thereby solving the system $Ax = b$, when A is SPD.

4.3. Anderson mixing

Anderson (1965) presented what he called an 'extrapolation algorithm' for accelerating sequences produced by fixed-point iterations. The method became well known in the physics literature as 'Anderson mixing', and later as 'Anderson acceleration' (AA) among numerical analysts. The following description of the algorithm introduces minimal changes to the notation adopted in the original paper. Anderson's scheme aimed at generalizing the simple mixing discussed earlier. It also starts with a mapping g that takes an input x into some output y, so that $y = g(x)$. The pair x, y is 'self-consistent' when the output and input values are the same, i.e. when $\|y - x\|_2 = 0$, or small enough in practice. In the following we define f to be the residual $f(x) = g(x) - x$. First, Anderson considered a pair consisting of an intermediate iterate $\bar{x}_j$ and an associated 'linear residual' $\bar{f}_j$:

$$\bar{x}_j = x_j + \sum_{i=[j-m]}^{j-1} \theta_i (x_i - x_j), \tag{4.11}$$

$$\bar{f}_j = f_j + \sum_{i=[j-m]}^{j-1} \theta_i (f_i - f_j). \tag{4.12}$$

Since $\bar{f}_j$ can be viewed as a linearized residual for $\bar{x}_j$, the idea is to determine the set of θ_i so as to minimize the 2-norm of $\bar{f}_j$:

$$\min_{\{\theta_i\}} \left\| f_j + \sum_{i=[j-m]}^{j-1} \theta_i (f_i - f_j) \right\|_2. \tag{4.13}$$

Once the optimal $\theta = [\theta_{[j-m]}, \ldots, \theta_{j-1}]^\top$ is found, the vectors $\bar{x}_j, \bar{f}_j$ are computed according to (4.11)–(4.12) and the next iterate is then defined as

$$x_{j+1} = \bar{x}_j + \beta \bar{f}_j. \tag{4.14}$$

In the original article β was dependent on j, but we removed this dependence for simplicity, as this also reflects common practice.

When comparing the Anderson scheme to Pulay mixing discussed in the next section, it becomes useful to rewrite equations (4.11)–(4.12) by defining $\theta_j = 1 - \sum_{i=[j-m]}^{j-1} \theta_i$. This leads to the mathematically equivalent equations

$$\bar{x}_j = \sum_{i=[j-m]}^{j} \theta_i x_i, \tag{4.15}$$

$$\bar{f}_j = \sum_{i=[j-m]}^{j} \theta_i f_i, \tag{4.16}$$

$$\{\theta_i\} = \text{argmin}\left\{ \left\| \sum_{i=[j-m]}^{j} \theta_i f_i \right\|_2 \quad \text{subject to} \quad \sum_{i=[j-m]}^{j} \theta_i = 1 \right\}. \tag{4.17}$$

Formulation (4.11)–(4.13) leads to a standard (unconstrained) least-squares problem to solve; see (4.13). It can be viewed as a straightforward alternative to the formulation (4.15)–(4.17), which requires solving a constrained optimization problem. The first formulation is still not an efficient one from the implementation point of view, mainly because the sets of vectors $f_i - f_j$ and $x_i - x_j$ for $i = [j-m], \ldots, j-1]$ must be computed at each step. An equivalent algorithm that avoids this will be seen in Section 6.

4.4. DIIS (Pulay mixing)

Direct inversion in the iterative subspace (DIIS) is a method introduced by Pulay (1980) to address the same acceleration needs as Anderson's method. It is well known as 'Pulay mixing' and widely used in computational chemistry. The method defines the new iterate in the form

$$x_{j+1} = \sum_{i=[j-m]}^{j} \theta_i g(x_i), \tag{4.18}$$

where the θ_i are parameters to be determined such that $\sum_{i=[j-m]}^{j} \theta_i = 1$. Defining the residual for iteration i as $f_i = g(x_i) - x_i$, the θ_i are selected to minimize the norm of the linearized residual, that is, they result from solving the following optimization problem:

$$\min \left\| \sum_{i=[j-m]}^{j} \theta_i f_i \right\|_2 \quad \text{subject to} \quad \sum_{i=[j-m]}^{j} \theta_i = 1. \tag{4.19}$$

In the original paper, the above problem was solved by using standard techniques based on Lagrange multipliers. However, an alternative approach is to invoke the

constraint and express one of the θ_i in terms of the others. Specifically, we can define

$$\theta_j = 1 - \sum_{i=[j-m]}^{j-1} \theta_i. \tag{4.20}$$

When this is substituted in the optimization problem (4.19) we obtain the same optimization problem (4.13) as for AA. In addition, assume that DIIS is applied to an iteration of the form (4.9) with γ replaced by β. In this case $g(x_i) = x_i + \beta f(x_i)$, and therefore the next iterate defined in (4.18) can be rewritten as

$$x_{j+1} = \sum_{i=1}^{j} \theta_i (x_i + \beta f(x_i)) = \sum_{i=[j-m]}^{j} \theta_i x_i + \beta \sum_{i=[j-m]}^{j} \theta_i f(x_i)) \equiv \bar{x}_j + \beta \bar{f}_j, \tag{4.21}$$

which is identical to Anderson's update in (4.14). Note also that the vectors f_i defined above are now $x_i - g(x_i) = \beta f_i$, so the solution to the problem (4.19) is unchanged. So the two schemes are equivalent when both are used to accelerate an iteration of the Richardson type (4.9). For this reason, the common term 'Anderson–Pulay mixing' is often used to refer to either method.

Later, Pulay (1982) considered improvements to the original scheme discussed above. This improved scheme consisted mainly in introducing a new SCF iteration, i.e. an alternative to the function g in our notation, which leads to better convergence. In other words the method is specifically targeted at SCF iterations. Pulay seemed unaware of Anderson's work, which preceded his by approximately 15 years. Indeed, neither of the two articles just mentioned cites Anderson's method.

Chupin, Dupuy, Legendre and Séré (2021) discuss the convergence of variable depth DIIS algorithms and show that these can lead to superior convergence and computationally more effective schemes.

5. Inexact and quasi-Newton approaches

Among other techniques that have been successfully used to accelerate fixed-point iterations, such as the ones in DFT, are those based on quasi-Newton (QN) approaches. One might argue whether or not it is legitimate to view these as 'acceleration techniques'. If the only restriction we put on acceleration methods is that they utilize a few of the previous iterates of the sequence, and that they apply the fixed-point mapping g to one or more vectors, then inexact Newton and quasi-Newton methods satisfy these requirements.

5.1. Inexact Newton

Many of the approaches developed for solving (1.2) are derived as approximations to Newton's approach, which is based on the local linear model around some current approximation x_j:

$$f(x_j + \Delta x) \approx f(x_j) + J(x_j)\Delta x, \tag{5.1}$$

where $J(x_j)$ is the Jacobian matrix at x_j. Newton's method determines $\delta = \Delta x_j = x_{j+1} - x_j$ at step j, to make the right-hand side on (5.1) equal to zero. This is achieved by solving the Newton linear system $J(x_j)\delta + f(x_j) = 0$. Inexact Newton methods (see e.g. Kelley 1995, Dembo, Eisenstat and Steihaug 1982, Brown and Saad 1990, among many references) compute a sequence of iterates in which the above Newton systems are solved approximately, typically by an iterative method. Starting from an initial guess x_0, the iteration proceeds as follows:

$$\text{solve} \quad J(x_j)\delta_j \approx -f(x_j), \tag{5.2}$$

$$\text{set} \quad x_{j+1} = x_j + \delta_j. \tag{5.3}$$

The right-hand side of the Newton system is $-f(x_j)$, and this is also the residual for the linear system when $\delta_j = 0$. Therefore in later sections we will define the residual vector $r_j \equiv -f(x_j)$.

Suppose that we invoke a Krylov subspace method for solving (5.2). If we set $J \equiv J(x_j)$, then the method will usually generate an approximate solution that can be written in the form

$$\delta_j = V_j y_j, \tag{5.4}$$

where V_j is an orthonormal basis of the Krylov subspace

$$\mathcal{K}_j = \text{Span}\{r_j, Jr_j, \ldots, J^{m-1}r_j\}. \tag{5.5}$$

The vector y_j represents the expression of the solution in the basis V_j. For example, if we use GMRES, or equivalently GCR (Eisenstat *et al.* 1983), then y_j becomes $y_j = (JV_j)^{\dagger}(-f(x_j))$. In essence, the inverse Jacobian is locally approximated by the rank m matrix:

$$B_{j,GMRES} = V_j(JV_j)^{\dagger}. \tag{5.6}$$

In inexact Newton methods the approximation just defined can be termed 'local', since it is only used for the jth step: once the solution is updated, the approximate inverse Jacobian (5.6) is discarded and the process will essentially compute a new Krylov subspace and related approximate Jacobian at the next iteration. This 'lack of memory' can be an impediment to performance. Herein lies an important distinction between these methods and quasi-Newton methods, whose goal is to compute approximations to the Jacobians, or their inverses, by an accumulative process, which utilizes the most recent iterates to gradually update these approximations.

5.2. *Quasi-Newton*

Standard quasi-Newton methods build a local approximation J_j to the Jacobian $J(x_j)$ progressively by using previous iterates. These methods require the relation (5.1) to be satisfied by the updated J_{j+1} which is built at step j. First the following *secant condition* is imposed:

$$J_{j+1}\Delta x_j = \Delta f_j, \tag{5.7}$$

where $\Delta f_j := f(x_{j+1}) - f(x_j)$. Such a condition will force the new approximation to be exact on the *pair* $\Delta x_j, \Delta f_j$ in the sense that the mapping J_{j+1} must transform Δx_j into Δf_j exactly. A second common requirement is the *no-change condition*:

$$J_{j+1}q = J_j q \quad \text{for all } q \quad \text{such that } q^\top \Delta x_j = 0. \tag{5.8}$$

In other words, there should be no new information from J_j to J_{j+1} along any direction q orthogonal to Δx_j. Broyden showed that there is a unique matrix J_{j+1} that satisfies both conditions (5.7) and (5.8), and it can be obtained by the update formula

$$J_{j+1} = J_j + (\Delta f_j - J_j \Delta x_j)\frac{\Delta x_j^\top}{\Delta x_j^\top \Delta x_j}. \tag{5.9}$$

Broyden's *second method* approximates the inverse Jacobian directly instead of the Jacobian itself. If G_j denotes this approximate inverse Jacobian at the jth iteration, then the secant condition (5.7) becomes

$$G_{j+1}\Delta f_j = \Delta x_j. \tag{5.10}$$

By minimizing $E(G_{j+1}) = \|G_{j+1} - G_j\|_F^2$ with respect to G_{j+1} subject to (5.10), one finds this update formula for the inverse Jacobian

$$G_{j+1} = G_j + (\Delta x_j - G_j \Delta f_j)\frac{\Delta f_j^\top}{\Delta f_j^\top \Delta f_j}, \tag{5.11}$$

which is also the only update satisfying both the secant condition (5.10) and the no-change condition for the inverse Jacobian

$$(G_{j+1} - G_j)q = 0 \quad \text{for all } q \perp \Delta f_j. \tag{5.12}$$

AA can be viewed from the angle of *multisecant* methods, i.e. block forms of the secant methods just discussed, in which we impose a secant condition on a whole set of vectors $\Delta x_i, \Delta f_i$ at the same time; see Section 6.7.1.

6. A detailed look at Anderson acceleration

Anderson (1965) hinted that he was influenced by the literature on 'extrapolation methods' of the type presented by Richardson and Shanks, and in fact he named his method the 'extrapolation method', although more recent terminology reserves this term to a different class of methods. There were several rediscoveries of the method, as well as variations in the implementations. In this section we will take a deeper look at AA and explore different implementations, variants of the algorithm, as well as theoretical aspects.

6.1. *Reformulating Anderson's method*

We begin by making a small adjustment to the notation in order to rewrite AA in a form that resembles that of extrapolation methods. This variation requires a simple

change of basis. As was seen in Section 4.3, Anderson used the basis vectors (in MATLAB colon notation)

$$d_i = f_j - f_i \quad \text{for} \quad i = [j-m]:\, j-1 \tag{6.1}$$

to express the vector added to f_j to obtain $\bar{f}_j$ as shown in (4.12), which becomes

$$\bar{f}_i = f_j - \sum_{i=[j-m]}^{j-1} \theta_i d_i. \tag{6.2}$$

We assume that the d_i do indeed form a basis. It is common in extrapolation methods to use forward differences, e.g. $\Delta f_i = f_{i+1} - f_i$ and $\Delta x_i = x_{i+1} - x_i$. These can be exploited to form an alternate basis consisting of the vectors $\Delta f_{i-1} = f_i - f_{i-1}$ for $i = [j-m]:\, j-1$ instead of the d_i. Note that the simple relations

$$\Delta f_i = (f_{i+1} - f_j) - (f_i - f_j) = d_i - d_{i+1}, \quad i = [j-m], \ldots, j-1, \tag{6.3}$$

$$\begin{aligned} f_j - f_i &= (f_j - f_{j-1}) + (f_{j-1} - f_{j-2}) + \cdots + (f_{i+1} - f_i) \\ &= \Delta f_{j-1} + \Delta f_{j-2} + \cdots + \Delta f_i, \quad i = [j-m], \ldots, j-1, \end{aligned} \tag{6.4}$$

allow us to switch from one basis representation to the other. The linear independence of one of these two sets of vectors implies the linear independence of the other. A similar transformation can also applied to express the vectors $x_j - x_i$ in terms of Δx_i and vice versa.

With this new notation at hand, we can rewrite AA as follows. Starting with an initial x_0 and $x_1 \equiv g(x_0) = x_0 + \beta f_0$, where $\beta > 0$ is a parameter, we define blocks of forward differences

$$\mathcal{X}_j = [\Delta x_{[j-m]} \ \cdots \ \Delta x_{j-1}], \quad \mathcal{F}_j = [\Delta f_{[j-m]} \ \cdots \ \Delta f_{j-1}]. \tag{6.5}$$

We will define $m_j = \min\{m, j\}$, which is the number of columns in $\mathcal{X}_j$ and $\mathcal{F}_j$. The least-squares problem (4.13) is translated to the new problem

$$\gamma^{(j)} = \text{argmin}_{\gamma \in \mathbb{R}^{m_j}} \|f_j - \mathcal{F}_j \gamma\|_2, \tag{6.6}$$

from which we get the vectors in equations (4.11), (4.12) and (4.14):

$$\bar{x}_j = x_j - \mathcal{X}_j\, \gamma^{(j)}, \tag{6.7}$$

$$\bar{f}_j = f_j - \mathcal{F}_j\, \gamma^{(j)}, \tag{6.8}$$

$$x_{j+1} = \bar{x}_j + \beta \bar{f}_j. \tag{6.9}$$

We have given this as Algorithm 4.

It is clear that the two methods are mathematically equivalent, as they both express the same underlying problem in two different bases. To better see the correspondence between the two in the simplest case when $m = \infty$ (in which case

Algorithm 4 Anderson acceleration AA(m)

Input: Function $f(x)$, initial guess x_0, window size m
Set $f_0 \equiv f(x_0)$;
$x_1 = x_0 + \beta_0 f_0$;
$f_1 \equiv f(x_1)$.
for $j = 1, 2, \ldots,$ until convergence **do**
    $\Delta x_{j-1} = x_j - x_{j-1} \quad \Delta f_{j-1} = f_j - f_{j-1}$
    Set $\mathcal{X}_j = [\Delta x_{[j-m]}, \ldots, \Delta x_{j-1}], \quad \mathcal{F}_j = [\Delta f_{[j-m]}, \ldots, \Delta f_{j-1}]$
    Compute $\gamma^{(j)} = \operatorname{argmin}_\gamma \|f_j - \mathcal{F}_j \gamma\|_2$
    Compute $x_{j+1} = (x_j - \mathcal{X}_j \gamma^{(j)}) + \beta(f_j - \mathcal{F}_j \gamma^{(j)})$
end for

$[j - m] = 0$) we can expand $f_j - \mathcal{F}_j \gamma$ and exploit (6.3):

$$\begin{aligned} f_j - \mathcal{F}_j \gamma &= f_j - \sum_{i=0}^{j-1} \gamma_{i+1}(f_{i+1} - f_i) \\ &= f_j - \sum_{i=0}^{j-1} \gamma_{i+1}[d_i - d_{i+1}] \quad \text{(note: } d_j \equiv 0\text{)} \\ &= f_j - \sum_{i=0}^{j-1} (\gamma_{i+1} - \gamma_i) d_i \quad \text{(with } \gamma_0 \equiv 0\text{)}. \end{aligned}$$

Thus a comparison with (6.2) shows that the optimal θ_i in the original Anderson algorithm can be obtained from the γ_i of the variant just presented by using the relation $\theta_i = \gamma_{i+1} - \gamma_i$ for $i = 0, \ldots, j-1$ with the convention that $\gamma_0 \equiv 0$. It is also easy to go in the opposite direction and express the γ_i from the θ_i of the original algorithm. In fact, a simple induction argument using the relations $\theta_i = \gamma_{i+1} - \gamma_i$ for $i \geq 0$ with $\gamma_0 \equiv 0$, will show that $\gamma_{i+1} = \sum_{j=0}^{i} \theta_j$ for $i \geq 0$.

The intermediate solution $\bar{x}_j$ in (6.7) can be interpreted as the minimizer of the linear model:

$$f(x_j - \mathcal{X}_j \gamma) \approx f(x_j) - \mathcal{F}_j \gamma, \tag{6.10}$$

where we use the approximation $f(x_j - \gamma_i \Delta x_i) \approx f(x_j) - \gamma_i \Delta f_i$. The method computes the minimizer of the linear model (6.10) and the corresponding optimal $\bar{x}_j$. The vector $\bar{f}_j$ is the corresponding linear residual. Anderson's method obtains this intermediate solution $\bar{x}_j$ and proceeds to perform a fixed-point iteration using the linear model represented by the pair $\bar{x}_j, \bar{f}_j$, as shown in (6.9).

From an implementation perspective, to compute the new iterate x_{j+1} as defined in (6.36) we need the pair x_j, f_j, the pair of arrays $\mathcal{X}_j, \mathcal{F}_j$, and β. We may write this as $x_{j+1} = \mathbf{AA}(x_j, \beta, \mathcal{X}_j, \mathcal{F}_j)$. The algorithm would first obtain $\gamma^{(j)}$ from solving (6.6), then compute $\bar{x}_j, \bar{f}_j$, and finally obtain the next iterate x_{j+1} from (6.9). In the restarted version of the algorithm, $\mathcal{X}_j, \mathcal{F}_j$ are essentially reset to empty arrays

every, say, k iterations. The case where all previous vectors are used, called the 'full window AA' (or just 'full AA'), corresponds to setting $m = \infty$ in Algorithm 4. The 'finite window AA' or 'truncated AA' corresponds to Algorithm 4, where m is finite. The parameter m is often termed the 'window size' or 'depth' of the algorithm.

6.2. *Classical implementations*

In the classical implementation of AA, the least-squares problems (6.6) or (4.13) are solved via the normal equations

$$(\mathcal{F}_j^\top \mathcal{F}_j)\gamma^{(j)} = \mathcal{F}_j^\top f_j, \tag{6.11}$$

where it is assumed that $\mathcal{F}_j$ has full rank. When the iterates approach convergence, the column vectors of $\mathcal{F}_j$ will tend to be close to zero or they may just become linearly dependent to within the available working accuracy. Solving the normal equations (6.11) in these situations will fail or be subject to severe numerical issues. In spite of this, the normal equation approach is rather common, especially when the window size m is small. The ideal solution to the problem from a numerical point of view is to resort to the singular value decomposition (SVD) and apply the truncated SVD (Golub and Van Loan 2013, § 5.5). However, this 'gold-standard' approach is expensive and has been avoided by practitioners. An alternative is to regularize the least-squares problem, replacing (6.11) with

$$(\mathcal{F}_j^\top \mathcal{F}_j + \tau I)\gamma^{(j)} = \mathcal{F}_j^\top f_j, \tag{6.12}$$

where τ is a regularization parameter. This compromise works reasonably well in practice and has the advantage of being inexpensive in terms of arithmetic and memory. Note that the memory cost of an approach based on normal equations is modest, mainly requiring us to keep the sets $\mathcal{F}_j, \mathcal{X}_j$, i.e. a total of $2m$ vectors of length n.

6.3. *Implementation with 'downdating QR'*

A numerically effective alternative to the normal equations is based on exploiting the QR factorization. Here we will focus on problem (6.6) of the formulation of AA discussed above. In principle the idea of using QR is straightforward. Start with a QR factorization of $\mathcal{F}_j$, which we write as $\mathcal{F}_j = QR$ with $Q \in \mathbb{R}^{n\times m_j}$, where we recall the notation $m_j = \min\{m, j\}$ and $R \in \mathbb{R}^{m_j\times m_j}$. Then obtain $\gamma^{(j)}$ by simply solving the triangular system $R\gamma^{(j)} = Q^\top f_j$. Such a trivial implementation runs into a practical difficulty, in that recomputing the QR factorization of $\mathcal{F}_j$ at each step is wasteful as it ignores the evolving nature of the block $\mathcal{F}_j$.

Assume at first that $j \le m - 1$. Then, at the end of the jth step represented in the main loop of Algorithm 4, one column is added to $\mathcal{F}_j$ (which has j columns) to obtain $\mathcal{F}_{j+1}$ (which has $j + 1 \le m$ columns). If $\mathcal{F}_j = QR$ and the new column

$v \equiv \Delta f_j$ is added, then we just need to carry out one additional step of the Gram–Schmidt process whereby this v is orthonormalized against the existing columns of Q, that is, we write $\hat{q} = v - Qh$ where $h = Q^\top v$ and then $q_{j+1} = \hat{q}/\rho$ where $\rho = \|\hat{q}\|_2$. Hence the new factorization is

$$\mathcal{F}_{j+1} = [\mathcal{F}_j, v] = [Q, q_{j+1}] \begin{pmatrix} R & h \\ 0 & \rho \end{pmatrix} \equiv \tilde{Q}\tilde{R}. \tag{6.13}$$

Assume now that $j > m - 1$. Then, to form $\mathcal{F}_{j+1}$ in the next step, a column v is added to $\mathcal{F}_j$ while the oldest one, Δf_{m_j}, must be discarded to create room for the new vector. Recall that the number of columns must stay $\leq m$. This calls for a different strategy based on 'downdating' the QR factorization, i.e. updating a QR factorization when a column of the original matrix is deleted.

To simplify notation in describing the process, we will remove indices and write our current $\mathcal{F}_j$ simply as $F = [v_1, v_2, \ldots, v_m]$ (m columns) and assume that F was previously factorized as $F = QR$. The aim is to build the QR factorization of $[v_2, v_3, \ldots, v_m]$ from the existing factors Q, R of F. Once this is done we will be in the same situation as the one where $j \leq m - 1$, which was treated above, and we can finish the process in the same way, as will be seen shortly. We write $F = [v_1, v_2, \ldots, v_m]$, $R = [r_1, r_2, \ldots, r_m]$ and denote the same matrices with their first columns deleted by

$$F_{(-)} = [v_2, v_3, \ldots, v_m], \quad H = [r_2, r_3, \ldots, r_m]. \tag{6.14}$$

The matrix H is of size $m \times (m-1)$ and has an upper Hessenberg form, i.e. $h_{ij} = 0$ for $i > j + 1$. The matrix $F_{(-)}$ satisfies

$$F_{(-)} = QH, \tag{6.15}$$

and our goal is to obtain a QR factorization of $F_{(-)}$ from this relation. For this we need to transform H into upper triangular form with the help of Givens rotation matrices (Golub and Van Loan 2013), as is often done. Givens rotations are matrices of the form

$$G(i,k,\theta) = \begin{pmatrix} 1 & \ldots & 0 & & \ldots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & \ldots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & \ldots & \vdots & \ldots & \vdots & \ldots & \vdots \\ 0 & \ldots & 0 & & \ldots & & 1 \end{pmatrix} \begin{matrix} \\ \\ i \\ \\ k \\ \\ \\ \end{matrix}$$

with $c = \cos\theta$ and $s = \sin\theta$. This matrix represents a rotation of angle $-\theta$ in the span of e_i and e_k. A rotation of this type is usually applied to transform some entry in the kth row of a matrix A to zero by an appropriate choice of θ. If $H = \{h_{ij}\}_{i=1:\,m, j=1:\,m-1}$, we can find a rotation $G_1 = G(1, 2, \theta_1)$ so that

$(G_1 H)_{2,1} = 0$. The values of c and s for this first rotation are

$$c = \frac{h_{11}}{\sqrt{h_{11}^2 + h_{12}^2}}, \quad s = \frac{h_{12}}{\sqrt{h_{11}^2 + h_{12}^2}}.$$

This is then followed by applying a rotation $G_2 = G(2, 3, \theta_2)$ so that $(G_2(G_1 H))_{3,2} = 0$, etc. Let Ω be the composition of these rotation matrices:

$$\Omega = G_{m-1} G_{m-2} \cdots G_2 G_1.$$

The process just described transforms H to an $m \times (m-1)$ upper triangular matrix with a zero last row:

$$\Omega H = \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix}.$$

Defining $\hat{Q} \equiv Q\Omega^\top \equiv [\hat{q}_1, \hat{q}_2, \ldots, \hat{q}_m]$ and $\hat{Q}^{(-)} \equiv [\hat{q}_1, \ldots, \hat{q}_{m-1}]$, we see that

$$F_{(-)} = QH = (Q\Omega^\top)\,(\Omega H) = \hat{Q} \times \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix} = \hat{Q}^{(-)} \hat{R}, \tag{6.16}$$

which is the factorization we were seeking. Once the factorization in (6.16) is available, we can proceed just as was done in the full window case ($j \leq m-1$) seen earlier to add a new vector v to the subspace, by updating the QR factorization of $[F_{(-)}, v]$ from that of $F_{(-)}$; see equation (6.13). Computing the downdated QR-factors in this way is much less expensive than recomputing a new QR factorization.

This procedure yields a pair of matrices Q_j, R_j such that $\mathcal{F}_j = Q_j R_j$ at each step j. The solution to the least-squares problem in line 8 is $\gamma^{(j)} = R_j^{-1} \eta^{(j)}$, where $\eta^{(j)} = Q_j^\top f_j$. Then, clearly, in line 9 of Algorithm 4 we have

$$f_j - \mathcal{F}_j \gamma^{(j)} = f_j - (Q_j R_j)\gamma^{(j)} = f_j - (Q_j R_j) R_j^{-1} \eta^{(j)} = f_j - Q_j \eta^{(j)}, \tag{6.17}$$

and thus lines 8–9 need to be replaced by

$$8a: \quad \eta^{(j)} = Q_j^\top f_j, \quad \gamma^{(j)} = R_j^{-1} \eta^{(j)}, \tag{6.18}$$

$$9a: \quad x_{j+1} = (x_j - \mathcal{X}_j \gamma^{(j)}) + \beta(f_j - Q_j \eta^{(j)}). \tag{6.19}$$

It is clear that the matrix Q_j can be computed 'in place' in the sense that the new Q_j can be computed and overwritten on the old one without requiring additional storage. Also, it is no longer necessary to store $\mathcal{F}_j$ in line 7. Instead we will perform a standard factorization where the vector $v = \Delta f_{j-1}$ is orthonormalized against those previous q_i that are saved. When $j > m-1$, a downdating QR factorization is performed to prepare the matrices $\hat{Q}^{(-)}, \hat{R}$ from which the updated QR is performed.

We will refer to this algorithm as 'AA-QR' if there is a need to specifically emphasize this particular implementation of AA. It will lead us to an interesting variant of AA to be discussed in Section 6.5.

6.4. *Simple downdating*

We now briefly discuss an alternative method for downdating the QR factorization. Returning to (6.15), we split the Hessenberg matrix H into its $(m-1)\times(m-1)$ lower block, which is triangular and which we denote by R, and its first row, which we denote by $h_1^\top$. With this, (6.15) can be rewritten as follows:

$$\begin{aligned} F_{(-)} &= q_1 h_1^\top + [q_2, q_2, \ldots, q_m]R \\ &= [Q_{(-)} + q_1 h_1^\top R^{-1}]R \\ &\equiv [Q_{(-)} + q_1 s^\top]R, \end{aligned} \tag{6.20}$$

where we have set $s^\top \equiv h_1^\top R^{-1}$. From here, there are two ways of proceeding. We can either get a QR factorization of $Q_{(-)} + q_1 s^\top$, which can then be retrofitted into (6.20), or we can provide some other orthogonal factorization that can be used to solve the least-squares problem effectively.

Consider the first approach. One way to get the QR factorization of $Q_{(-)}+q_1 s^\top$ is via the Cholesky factorization of the matrix $[Q_{(-)}+q_1 s^\top]^\top[Q_{(-)}+q_1 s^\top]$. Observing that the columns of $Q_{(-)}$ are orthonormal and orthogonal to q_1, we see that

$$[Q_{(-)} + q_1 s^\top]^\top[Q_{(-)} + q_1 s^\top] = I + ss^\top. \tag{6.21}$$

As it turns out, matrices of the form $I + ss^\top$ lead to an inexpensive Cholesky factorization due to a nice structure that can be exploited.[3] From this factorization, say $I + ss^\top = GG^\top$ where G is lower triangular, we get the updated factorization

$$F_{(-)} = (Q_{(-)} + q_1 s^\top)R \tag{6.22}$$

$$= \underbrace{[(Q_{(-)} + q_1 s^\top)G^{-T}]}_{Q}\ \underbrace{[G^\top R]}_{R} \equiv QR. \tag{6.23}$$

The Q-factor defined above can be verified to be indeed an orthogonal matrix while the R-factor is clearly upper triangular. Implementation and other details are omitted.

The second approach is a simplified, possibly more appealing, procedure. Here we no longer rely on a formal QR factorization, but on a factorization that nonetheless exploits an orthogonal factor. Starting from (6.22), we now write

$$F_{(-)} = \underbrace{[(Q_{(-)} + q_1 s^\top)(I + ss^\top)^{-1/2}]}_{Q}\ \underbrace{(I + ss^\top)^{1/2}R]}_{S} \equiv QS. \tag{6.24}$$

[3] In a nutshell, the entries below the main diagonal of the jth column of the L matrix in the LDLT decomposition of the matrix $I+ss^\top$ are a constant times the entries $j+1:n$ of s. This observation leads to a Cholesky factorization that is inexpensive and easy to compute and to exploit.

Note in passing that the product $Q(I + ss^\top)^{1/2}$ is just the *polar decomposition* (Golub and Van Loan 2013) of the matrix $Q_{(-)} + q_1 s^\top$. Here too the structure of $I + ss^\top$ leads to simplifications, in this case for the terms in the fractional powers of $I + ss^\top$. Indeed, if we set $\lambda = 1 + s^\top s$, then it can be shown that

$$(I + ss^\top)^{1/2} = I + \alpha ss^\top, \quad \text{with } \alpha = \frac{1}{1 + \sqrt{\lambda}}, \tag{6.25}$$

$$(I + ss^\top)^{-1/2} = I - \beta ss^\top, \quad \text{with } \beta = \frac{1}{\lambda + \sqrt{\lambda}}. \tag{6.26}$$

Noting that $Q^\top Q = I$, we wind up with a factorization of the form $F_{(-)} = QS$, where Q has orthonormal columns but $S = (I + ss^\top)^{1/2} R$ is not triangular. It is possible to implement this by keeping the matrix S as a square matrix or in factored form in order to exploit its inverse, which is $S^{-1} = R^{-1}(I - \beta ss^\top)$, where β is given in (6.26). The next step, adding a vector to the system, can be processed as in a standard QR factorization (see (6.13)), where the block R is to be replaced by S.

These two simple alternatives to the Givens-based downdating QR do not seem to have been considered in the literature. Their main merit is that they focus more explicitly on the consequence of deleting a column and show what remedies can be applied. The deletion of a column leads to the QR-like factorization (6.22) of $F_{(-)}$. This resembles a QR factorization but the Q-part, namely $Q_{(-)} + q_1 s^\top$, is not orthogonal. The standard remedy is to proceed with a downdating QR factorization that obtains QR-factors from (6.15) instead of (6.20). Instead, the remedies outlined above proceed from (6.20), from which either the QR factorization of this matrix or its polar decomposition are derived. In both cases, the results are then retrofitted into (6.22) to obtain a corrected decomposition, with an orthogonal Q-factor.

6.5. The truncated Gram–Schmidt (TGS) variant

There is an appealing alternative to AA-QR worth considering: bypass the downdating QR, and just use the orthogonal factor Q_j obtained from a truncated Gram–Schmidt (TGS) orthogonalization. This means that a new vector q_j is obtained by orthonormalizing Δf_{j-1} against the previous q_i that are saved, and when we delete a column we just omit the adjustment consisting of the downdating QR process. Without this adjustment the resulting factors Q_j, R_j are no longer a QR factorization of $\mathcal{X}_j$ when a truncation has taken place, i.e. when $j > m$. The solution to the least-squares problem now uses Q_j instead of $\mathcal{F}_j$, that is, we minimize $\|f_j - Q_j\gamma\|_2$, ending with $\gamma^{(j)} = Q_j^\top f_j$. In the limited window case, the resulting least-squares approximation to f_j, i.e. $Q_j\gamma^{(j)}$, is no longer equal to the one resulting from the downdating QR approach, which solves the original LS problem (4.13) with the set $\mathcal{F}_j$ by exploiting its QR factorization. In Anderson acceleration with truncated Gram–Schmidt (AA-TGS), *the resulting matrix Q_j is not required to be the Q-factor of $\mathcal{F}_j$ and so the range of $\mathcal{F}_j$ is different from that of Q_j*. Therefore the method is *not equivalent* to AA in the finite window case.

We can write the orthogonalization process at the jth step as

$$q_j = \frac{1}{s_{jj}}\left[\Delta f_{j-1} - \sum_{i=[j-m+1]}^{j-1} s_{ij} q_i\right]. \tag{6.27}$$

Here the scalars $s_{ij}, i = [j-m+1], \ldots, j-1$ are those utilized in a modified Gram–Schmidt process, and s_{jj} is a normalizing factor so that $\|q_j\|_2 = 1$.

With $m_j \equiv \min\{m, j+1\}$, define the $m_j \times m_j$ upper triangular matrix $S_j = \{s_{ik}\}_{i=[j-m+1]:j, k=[j-m+1]:j}$ resulting from the orthogonalization process in (6.27). Having selected the set of columns to use in place of $\mathcal{F}_j$, the question now is how to compute the solution x_{j+1}, that is, how do we change line 9 of Algorithm 4? One is tempted to take as a model equations (6.18)–(6.19) of AA-QR, where the matrix R_j in (6.18) is replaced by[4] S_j. However, the relation $\mathcal{F}_j = Q_j R_j$ is only valid in the full window case, so the relation $f_j - \mathcal{F}_j R_j^{-1}\eta^{(j)} = f_j - Q_j\eta^{(j)}$ in (6.17) no longer holds, and thus we cannot write $\bar{x}_j$ in the form $\bar{x}_j = x_j - \mathcal{X}_j R_j^{-1}\eta^{(j)}$. The solution is to make use of the basis U_j that is defined from the set of Δx_i in the same way that Q_j is defined from the Δf_i. Thus we compute the jth column of U_j by the same process we applied to obtain q_j, namely

$$u_j = \frac{1}{s_{jj}}\left[\Delta x_{j-1} - \sum_{i=[j-m+1]}^{j-1} s_{ij} u_i\right], \tag{6.28}$$

where the scalars s_{ij} are the same as those utilized to obtain q_j in (6.27). With this, the relations (6.7)–(6.9) are replaced by

$$\bar{x}_j = x_j - U_j\eta^{(j)}, \quad \bar{f}_j = f_j - Q_j\eta^{(j)}, \quad x_{j+1} = \bar{x}_j + \beta\bar{f}_j. \tag{6.29}$$

The procedure is sketched as Algorithm 5. Let us examine the algorithm and compare it with the downdating QR version seen in Section 6.3. When $j \le m$ (full window case), the i loop starting in line 6, begins at $i = 0$, and the block in lines 6–12 essentially performs a Gram–Schmidt QR factorization of the matrix $\mathcal{F}_j$, while also enforcing identical operations on the set $\mathcal{X}_j$. A result of the algorithm is that

$$\mathcal{F}_j = Q_j S_j, \quad \mathcal{X}_j = U_j S_j. \tag{6.30}$$

The above relations are not valid when $j > m$. In particular, the subspace spanned by Q_j is no longer the span of $\mathcal{F}_j$. We saw how to deal with this issue in Section 6.3 in order to recover a QR factorization for $\mathcal{F}_j$ from a QR downdating process. AA-TGS provides an alternative solution by relaxing the requirement of having to use a QR factorization of $\mathcal{X}_j$.

To better understand the process, we examine what happens specifically when $j = m+1$, focusing on the set of q_i. We adopt the same abbreviated notation as in

[4] Recall that for convenience the indexing of the columns in Q and other arrays in Section 6.3 start at 1 instead of 0.

Algorithm 5 AA-TGS(m)

Input: Function $f(x)$, initial guess x_0, window size m
Set $f_0 \equiv f(x_0)$, $\quad x_1 = x_0 + \beta_0 f_0$, $\quad f_1 \equiv f(x_1)$
for $j = 1, 2, \ldots,$ until convergence **do**
  $u \coloneqq \Delta x = x_j - x_{j-1}$
  $q \coloneqq \Delta f = f_j - f_{j-1}$
  for $i = [j - m + 1], \ldots, j - 1$ **do**
    $s_{ij} \coloneqq (q, q_i)$
    $u \coloneqq u - s_{ij} u_i$
    $q \coloneqq q - s_{ij} q_i$
  end for
  $s_{jj} = \|q\|_2$
  $q_j \coloneqq q / s_{jj}$, $\quad u_j \coloneqq u / s_{jj}$
  Set $Q_j = [q_{[j-m+1]}, \ldots, q_j]$, $\quad U_j = [u_{[j-m+1]}, \ldots, u_j]$
  Compute $\eta^{(j)} = Q_j^\top f_j$
  $x_{j+1} = (x_j - U_j \eta^{(j)}) + \beta_j (f_j - Q_j \eta^{(j)})$
  $f_{j+1} = f(x_{j+1})$
end for

Section 6.3, and in particular the indexing in arrays Q and S starts at 1 instead of 0. Before the orthogonalization step we have the QR factorization $\mathcal{F}_m = Q_m S_m$. Dropping the oldest (first) column is captured by equations (6.14) and (6.15). We rewrite (6.15) as follows:

$$F_{(-)} = QH = [q_1, q_2, ..., q_m]H = q_1 h_1^\top + [q_2, q_3, \ldots, q_m] S_{(-)}.$$

As before, H is the $m \times (m-1)$ Hessenberg matrix obtained from the upper triangular matrix S_m by deleting its first column. The row vector $h_1^\top$ is the first row of H (first row of S_m, omitting its first entry). The matrix $S_{(-)}$ is the $(m-1) \times (m-1)$ upper triangular matrix obtained from S_m by deleting its first row and its first column. Thus, if we let $Q_{(-)} = [q_2, q_3, \ldots, q_m]$ as before, we obtain a reduced QR factorization but it is for a different matrix, that is,

$$F_{(-)} - q_1 h_1^\top = Q_{(-)} S_{(-)}. \tag{6.31}$$

After the truncation, the new vector $v_{m+1} = \Delta f_m$ is orthonormalized against $q_2, q_3, \ldots, q_m$, leading to the next vector q_{m+1}. Then we can write

$$\begin{aligned} [F_{(-)} - q_1 h_1^\top, v_{m+1}] &= [Q_{(-)}, q_{m+1}] \times \begin{bmatrix} S_{(-)} & s_{2:m,m+1} \\ 0 & s_{m+1,m+1} \end{bmatrix} \\ &\equiv Q_{m+1} S_{m+1}. \end{aligned} \tag{6.32}$$

Therefore at step $j = m + 1$ the pair of matrices Q_{m+1}, S_{m+1} *produce a QR factorization of a rank-one perturbation of the matrix* $\mathcal{F}_{m+1}$.

Herein lies the only difference between the two methods: the downdating QR enforces the relation $\mathcal{F}_j = Q_j R_j$ by correcting the relation (6.31) into a valid factorization of $F_{(-)}$ before proceeding. We saw in Sections 6.3 and 6.4 how this can be done. This ensures that we obtain the same solution as with AA. In contrast, AA-TGS simplifies the process by not insisting on having a QR factorization of $\mathcal{F}_j$. Instead, it exploits a QR factorization of a modified version of $\mathcal{F}_j$; see (6.32) for the case $j = m + 1$. Note that when $j > m + 1$ the rank-one modification on the left-hand side of (6.32) becomes a sum of rank-one matrices.

Let us now consider the full window case, i.e. the situation $j \leq m$. It is easy to see that in this case the subspaces spanned by $\mathcal{F}_j$ and Q_j are identical, and in this situation the iterates x_{j+1} resulting from AA and AA-TGS will be the same. In particular, when $m = \infty$ this will always be the case. We will state this mathematical equivalence of the two algorithms in the following proposition.

Proposition 6.1. Assuming that they start from the same initial guess x_0, AA-TGS (∞) and AA(∞) return the same iterates at each iteration, in exact arithmetic. In addition, they also break down under the same condition.

The proof is straightforward and relies on the equality $\mathrm{Span}\{Q_j\} = \mathrm{Span}\{\mathcal{X}_j\}$ for all j. Although rather trivial, this property is worth stating explicitly because it will help us simplify our analysis of AA in the full window case. It is clear that we can also state a more general result for the restarted versions of the algorithms.[5]

We end this section with an important addition to the AA-TGS algorithm whose goal is to circumvent some numerical stability issues. The two recurrences induced by equations (6.27)–(6.28) are linear recurrences that can lead to instability. A mechanism must be added to monitor the behaviour of the above sequence with the help of a scalar sequence whose numerical behaviour imitates that of the vector sequences. This will help prevent excessive growth of rounding errors by restarting the process when deemed necessary. A process of this type was developed in Tang *et al.* (2024). Readers are referred to the article for details.

6.6. *Numerical illustration*

We will illustrate a few of the methods seen so far in this paper with two examples. The first example is usually viewed as an easy problem to solve because its level of nonlinearity can be characterized as mild. The second is an optimization problem in molecular physics with highly nonlinear coefficients.

[5] Restarting can be implemented for any of the algorithms seen in this article. By restarting, we mean that at every k iterations the algorithm starts anew with x_0 replaced by the most recent approximation computed. The parameter k is called the 'restart dimension', or 'period'. Other restarting strategies are not periodic, and instead restart when deemed necessary by the numerical behaviour of the iterates.

6.6.1. The Bratu problem

The Bratu problem appears in modelling thermal combustion, radiative heat transfer and thermal reaction, among other applications; see e.g. Mohsen (2014) and Jacobsen and Schmitt (2002) for references. It consists in the following nonlinear elliptic PDE with Dirichlet boundary conditions on an open domain Ω:

$$\begin{aligned} \Delta u + \lambda e^u &= 0 \ \text{ in } \ \Omega, \\ u(x, y) &= 0 \ \text{ for } (x, y) \in \partial\Omega. \end{aligned}$$

Here λ is a parameter and there is a solution only for values of λ in a certain interval. We set λ to the value $\lambda = 0.5$ and define the domain Ω to be the square $\Omega = (0, 1) \times (0, 1)$. Discretization with central differences using 100 interior points in each direction results in a system of nonlinear equations $f(x) = 0$, where f is a mapping from $\mathbb{R}^n$ to itself, with $n = 10\,000$.

The first step in applying acceleration techniques to solve the problem is to formulate an equivalent fixed-point iteration of the form

$$g(x) = x - \mu f(x). \tag{6.33}$$

The reader may have noted the negative sign used for μ instead of the positive sign seen in earlier formulas for the mixing scheme (4.9). In earlier notation $f(x)$ represented the 'residual', i.e. typically the negative of the gradient of some function ϕ, instead of the gradient as is the case here. What value of μ should we use? Noting that the Jacobian of g is

$$\frac{\partial g}{\partial x}(x) = I - \mu \frac{\partial f}{\partial x}(x),$$

a rule of thumb, admittedly a vague one, is that a small value is needed only when we expect the Jacobian of f to have large values. For all experiments dealing with the Bratu problem, we will set $\mu = 0.1$.

6.6.2. Molecular optimization with Lennard-Jones potential

The goal of geometry optimization is to find atom positions that minimize total potential energy as described by a certain potential. The Lennard-Jones (LJ) potential has a long history and is commonly used in computational chemistry; see Kittel (1986, p. 61). Its aim is to represent both attraction and repulsion forces between atoms by the inclusion of two terms with opposite signs:

$$E = \sum_{i=1}^{N} \sum_{j=1}^{i-1} 4 \times \left[\frac{1}{\|x_i - x_j\|_2^{12}} - \frac{1}{\|x_i - x_j\|_2^{6}} \right]. \tag{6.34}$$

Each x_i is a three-dimensional vector whose components are the coordinates of the location of atom i. A common problem is to start with a certain configuration and then optimize the geometry by minimizing the potential starting from that position. Note that the resulting position is not a global optimum but a local minimum

around some initial configuration. In this particular example we simulate an Argon cluster by taking the initial position of the atoms to consist of a perturbed initial face-centred cubic structure (Meyer, Barrett and Haasen 1964). We took three cells per direction, resulting in 27 unit cells. FCC cells include four atoms each, so we end up with a total of 108 atoms. The problem can be challenging due to the high powers in the potential.

Instead of a nonlinear system of equations as was the case for the Bratu problem, we now need to minimize $E(\{x_i\})$. The gradient of E with respect to atom positions can be readily computed. If we let x denote the vector that contains the coordinates of all atoms, we let $f(x)$ denote this gradient, i.e. $f(x) = \nabla E(x)$. The associated fixed-point mapping is again of the form (6.33), but this time we will need to take a much smaller value for μ, namely $\mu = 0.0001$. Larger values of μ often result in unstable iterates, overflow, or convergence to a non-optimal configuration. Note that we are seeking a local minimum and as such we do need to verify for each run that the scheme being tested converges to the correct optimal configuration, in this case a configuration that has the potential $E_{opt} = -579.4639$.

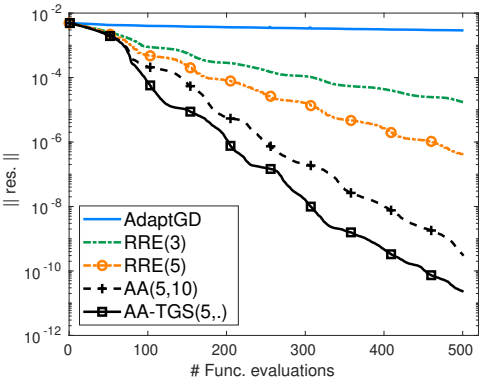
6.6.3. Gradient descent, RRE, Anderson, and Anderson-TGS

We will illustrate four algorithms for the two problems discussed above. The first is a simple adaptive gradient descent algorithm of the form $x_{j+1} = x_j - \mu f(x_j)$, where μ is set adaptively by a very simple scheme: if the norm of $f(x_j)$ increases, multiply μ by 0.3, and if it decreases, multiply μ by 1.05. The initial μ is as defined earlier: $\mu = 0.1$ for Bratu and $\mu = 0.0001$ for LJ. We will call this scheme `adaptGD`. The second scheme tested is a restarted version of the RRE algorithm seen in Section 2.6. If m is the restart dimension then the scheme computes an accelerated solution y_m using $x_0, \ldots, x_{m+1}$, and then sets x_0 to be equal to y_m and the algorithm is continued from this x_0. It is interesting to note that we use $x_0, \ldots, x_m, x_{m+1}$ to compute y_m, which is an update to x_m and not x_{m+1}; see (2.39). Thus in effect x_{m+1} is only used to obtain the optimal γ in (2.39). Anderson acceleration takes care of this by the extra approximate fixed-point step (6.9). We can perform a similar step for RRE, and it will translate to

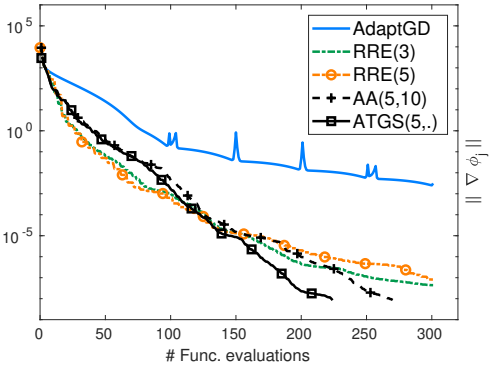
$$x_{m+1} = y_m + \beta \bar{f}_m, \tag{6.35}$$

where $\bar{f}_m = \Delta x_m + \Delta X_0 \gamma$ is the linear residual. This modification is implemented with $\beta = 1$ in the experiments that follow. In addition to the baseline adaptive GD, we test RRE(3), RRE(5) and Anderson(5,10) for both problems. We also tested Anderson-TGS(5,.) with the automatic restarting strategy briefly mentioned at the end of Section 6.5 and described in detail in Tang *et al.* (2024). We should point out that RRE(5), Anderson(5,10) and Anderson-TGS(5,.) all use roughly the same amount of memory.

Figure 6.1 shows the results obtained by these methods for both problems. Here AA(5,10) stands for AA with window size 5 and restart dimension $k = 10$. RRE(m) is the restarted RRE procedure described above with restart dimension m,



(a)



(b)

Figure 6.1. Comparison of five different methods for the Bratu problem (a) and the Lennard-Jones optimization problem (b).

where $m = 3$ in the first experiment with RRE and $m = 5$ in the second. The plots show the norm of the residual (for Bratu) and the norm of the gradient (for LJ) versus the number of function evaluations. All methods start from the same random initial guess and are stopped when either the residual norm decreases by $tol = 10^{-12}$ or the number of function evaluations exceeds a certain maximum (500 for the Bratu problem, 300 for Lennard-Jones). As can be seen, for the Bratu problem, Anderson and its TGS variant both yield a good improvement relative to the simpler RRE schemes. For the Lennard-Jones problem, in contrast, the performance of AA(5,10) is close to that of RRE. Surprisingly, RRE(3), which uses much less memory, does quite well for this example. In both test problems, AA-TGS outperforms the standard Anderson algorithm by a moderate margin. The standard Anderson scheme discussed in these experiments is based on the AA-QR (AA with QR downdating) implementation discussed in Section 6.3.

6.7. *Theory*

Around the year 2009, Anderson acceleration started to be noticed in the linear algebra community, no doubt owing to its simplicity and its success in dealing with a wide range of problems. Fang and Saad (2009) discussed *multisecant methods*, first introduced as 'generalized Broyden methods' by Vanderbilt and Louie (1984). These methods can be called block secant methods in which rank-one updates of the form (5.9) or (5.11) are replaced by rank-m updates. As it turns out, AA is a multisecant method, and this specific link was first unravelled in Eyert (1996) and discussed further in Fang and Saad (2009). A number of other results were subsequently shown. Among these is an equivalence between Krylov methods and Anderson in the linear case as well as convergence studies for nonlinear sequences. This goal of this section is to summarize these results.

6.7.1. *AA as a multisecant approach*

Acceleration methods, such as AA, do not aim at solving a system like (1.2) directly. As was seen in Section 4.3, their goal is to accelerate a given fixed-point iteration of the form (4.4). The method implicitly expresses an approximation to the Jacobian via a secant relation which puts $\mathcal{F}_j$ in correspondence with $\mathcal{X}_j$. Roughly speaking, AA develops some approximation to a Jacobian J that satisfies a secant condition of the form $\mathcal{F}_j \approx J\mathcal{X}_j$. The classic text by Ortega and Rheinbolt (1970, pp. 204–205) already mentions AA as a form of quasi-Newton approach. At the time, however, Ortega and Rheinbolt viewed the method negatively, possibly due to its potential for numerically unstable behaviour.

As seen in Section 5, Broyden-type methods replace Newton's iteration, $x_{j+1} = x_j - J(x_j)^{-1} f_j$, with something like $x_{j+1} = x_j - G_j f_j$, where G_j approximates the inverse of the Jacobian $J(x_j)$ via the update formula

$$G_{j+1} = G_j + (\Delta x_j - G_j \Delta f_j) v_j^\top$$

in which v_j is defined in several different ways; see Fang and Saad (2009) for details. AA belongs to the related class of *multisecant methods*. Indeed, the approximation (6.9) can be written as

$$\begin{aligned} x_{j+1} &= x_j + \beta f_j - (\mathcal{X}_j + \beta \mathcal{F}_j)\, \theta^{(j)} && (6.36)\\ &= x_j - [-\beta I + (\mathcal{X}_j + \beta \mathcal{F}_j)(\mathcal{F}_j^\top \mathcal{F}_j)^{-1} \mathcal{F}_j^\top] f_j && (6.37)\\ &\equiv x_j - G_j f_j, \quad \text{with } G_j \equiv -\beta I + (\mathcal{X}_j + \beta \mathcal{F}_j)(\mathcal{F}_j^\top \mathcal{F}_j)^{-1} \mathcal{F}_j^\top. && (6.38) \end{aligned}$$

Thus G_j can be seen as an update to the (approximate) inverse Jacobian $G_{[j-m]} = -\beta I$ by the formula

$$G_j = G_{[j-m]} + (\mathcal{X}_j - G_{[j-m]} \mathcal{F}_j)(\mathcal{F}_j^\top \mathcal{F}_j)^{-1} \mathcal{F}_j^\top. \quad (6.39)$$

It can be shown that the approximate inverse Jacobian G_j is the result of minimizing

$\|G_j + \beta I\|_F$ under the *multisecant condition* of type II[6]

$$G_j \mathcal{F}_j = \mathcal{X}_j. \tag{6.40}$$

This link between AA and Broyden multisecant-type updates was first unravelled by Eyert (1996) and expanded upon in Fang and Saad (2009). Thus the method is in essence what we might call a 'block version' of Broyden's second update method, whereby a rank-m, instead of rank-one, update is applied at each step.

In addition, we also have a multisecant version of the no-change condition (5.12). This is just a block version of the no-change condition equation (5.12) as represented by equation (15) in Fang and Saad (2009), which stipulates that

$$(G_j - G_{[j-m]})q = 0 \quad \text{for all } q \perp \text{Span}\{\mathcal{F}_j\} \tag{6.41}$$

provided we define $G_{[j-m]} = 0$.

The essence of AA is that it approximates $f(x_j - \Delta X_j\ \gamma)$ by $f(x_j) - \Delta F_j\ \gamma$. If J_j is the Jacobian at x_j, we are approximating $J_j\ \Delta X_j$ by ΔF_j and aim to make $f(x_j - \Delta X_j\ \gamma)$ small by selecting γ so that its linear approximation $f(x_j) - \Delta F_j\ \gamma$ has a minimal norm. Thus, just like quasi-Newton approaches, the method also aims to exploit earlier iterates in order to approximate J_j, or its action on a vector or a set of vectors. The main approach relies on two sets of vectors, which we call $P_j = [p_{[j-m+1]}, p_{[j-m+1]+1}, \ldots, p_j]$ and $V_j = [v_{[j-m+1]}, v_{[j-m+1]+1}, \ldots, v_j]$ in this paper, with the requirement that

$$J(x_j)p_j \approx v_j. \tag{6.42}$$

These 'secant' conditions establish a correspondence between the range of P_j and the range of V_j and are at the core of any multisecant method.

6.7.2. *Interpretations of AA*

Anderson's original algorithm can be interpreted from a number of different perspectives. He acknowledged being inspired by work on extrapolation methods similar to those discussed in Section 2. However, the method he introduced does not fit the definition of an extrapolation technique according to the terminology we use in this paper.

Equation (6.9) resembles a simple Richardson iteration (see (3.2)) applied to the intermediate iterate $\bar{x}_j$ and its (linear) residual $\bar{f}_j$. Therefore the first question we could address is what $\bar{x}_j$ and $\bar{f}_j$ represent. AA starts by computing an intermediate approximation solution that is denoted by $\bar{x}_j$. The approximation $\bar{x}_j$ is just a member of the affine space $x_j + \text{Span}\{\mathcal{X}_j\}$. (In the equivalent initial presentation of AA, it was of the form given by (4.11).) We would normally write any vector on this space as $x_j + \mathcal{X}_j\gamma$, where γ is an arbitrary vector in $\mathbb{R}^{m_j}$, where $m_j = \min\{j, m\}$

[6] Type I Broyden conditions involve approximations to the Jacobian, while type II conditions deal with the inverse Jacobian.

is the number of columns of $\mathcal{X}_j$. Anderson changed the sign of the coefficient γ, so instead he considered vectors of the form

$$x(\gamma) = x_j - \mathcal{X}_j\gamma. \tag{6.43}$$

Ideally, we would have liked to compute a coefficient vector γ that minimizes the norm $\|f(x(\gamma))\|_2$. This is computable by a line search but at a high cost, so instead Anderson exploits the linear model around x_j:

$$f(x(\gamma)) \equiv f(x_j - \mathcal{X}_j\gamma) \approx f(x_j) - \mathcal{F}_j\gamma = f_j - \mathcal{F}_j\gamma. \tag{6.44}$$

The above expression is therefore the *linear residual* at x_j for vectors of the form (6.43). The optimal γ that we denoted by $\gamma^{(j)}$ is precisely what is computed in (6.6). Therefore $\bar{x}_j$ is just *the vector of the form* (6.43) *that achieves the smallest linear residual.*

An important observation here is that we could have considered the new sequence $\{\bar{x}_j\}_{j=0,1,\ldots}$ by itself. Remarkably, each vector $\bar{x}_j$ is just a linear combination of the previous x_i, so it represents an *extrapolated sequence* of the form (2.13). In fact the vector $\bar{x}_j$ can be seen to be identical to the vector produced by the RRE algorithm seen in Section 2.6. Computing this extrapolated sequence by itself, *without mixing it with the original iterates*, will be identical to applying the RRE procedure to the x_i. It gets us closer to the solution by combining previous iterates, but we can do better. Since $\bar{x}_j$ is likely to be a better approximation than x_j, a reasonable option would be to define the next iterate as a fixed-point iteration based at $\bar{x}_j$:

$$x_{j+1}^{(+)} = \bar{x}_j + \beta f(\bar{x}_j). \tag{6.45}$$

This, however, would require an additional function evaluation. Therefore AA replaces $f(\bar{x}_j) = f(x_j - \mathcal{X}_j\gamma^{(j)})$ with its linear approximation given in (6.44), which is just $\bar{f}_j$, resulting in the Anderson update given by (6.9). Thus Anderson acceleration can be viewed as a process that intermingles one step of RRE applied to previous iterates with one linearized gradient descent of the form (6.9). An alternative would be to restart – say every k RRE steps – and reset the next iterate to be the linearized update (6.9). We tested a technique of this type in the experiment shown in Section 6.6.

6.7.3. Linear case: Links with Krylov subspace methods

In this section we consider the case when the problem is linear, and set $f(x) = b - Ax$ (note the sign difference from earlier notation). We assume that A is non-singular. In this situation we have

$$\mathcal{F}_j = -A\mathcal{X}_j. \tag{6.46}$$

The following lemma shows that that the matrix U_j resulting from Algorithm 5 is a basis of the Krylov subspace $\mathcal{K}_j(A, f_0)$, and that under mild conditions Q_j, U_j satisfy the same relation as $\mathcal{F}_j, \mathcal{X}_j$ in (6.46) for AA-TGS(∞).

Lemma 6.1. Assume A is invertible and $f(x) = b - Ax$. If Algorithm 5 used to solve $f(x) = 0$ with $m = \infty$ does not break down at step j, then the system U_j forms a basis of the Krylov subspace $\mathcal{K}_j(A, f_0)$. In addition, the orthonormal system Q_j built by Algorithm 5 satisfies $Q_j = -AU_j$.

Proof. We first prove $Q_j = -AU_j$ by induction. When $j = 1$, we have $q_1 = (f_1 - f_0)/s_{11} = -Au_1$. Assume $Q_{j-1} = -AU_{j-1}$. Then we have

$$\begin{aligned} s_{jj}q_j &= (f_j - f_{j-1}) - \sum_{i=1}^{j-1} s_{ij}q_i \\ &= -A(x_j - x_{j-1}) - \sum_{i=1}^{j-1} s_{ij}(-Au_i) \\ &= -A\left[(x_j - x_{j-1}) - \sum_{i=1}^{j-1} s_{ij}u_i\right] \\ &= s_{jj}(-Au_j). \end{aligned}$$

Thus, since $s_{jj} \neq 0$ we get $q_j = -Au_j$ and therefore $Q_j = -AU_j$, completing the induction proof.

Next, we prove by induction that U_j forms a basis of $\mathcal{K}_j(A, f_0)$. It is more convenient to prove inductively that, for each $i \leq j$, U_i forms a basis of $\mathcal{K}_i(A, f_0)$. The result is true for $j = 1$ since we have $u_1 = (x_1 - x_0)/s_{11} = \beta_0 f_0/s_{11}$. Now let us assume the property is true for $j - 1$, that is, for each $i = 1, 2, \ldots, j - 1$, U_i is a basis of the Krylov subspace $\mathcal{K}_i(A, f_0)$. Then we have

$$\begin{aligned} s_{jj}u_j &= (x_j - x_{j-1}) - \sum_{i=1}^{j-1} s_{ij}u_i \\ &= -U_{j-1}\theta_{j-1} + \beta_{j-1}(f_{j-1} - Q_{j-1}\theta_{j-1}) - \sum_{i=1}^{j-1} s_{ij}u_i \\ &= -U_{j-1}\theta_{j-1} + \beta_{j-1}f_{j-1} - \beta_{j-1}Q_{j-1}\theta_{j-1} - \sum_{i=1}^{j-1} s_{ij}u_i \\ &= \beta_{j-1}f_{j-1} - U_{j-1}\theta_{j-1} + \beta_{j-1}AU_{j-1}\theta_{j-1} - \sum_{i=1}^{j-1} s_{ij}u_i. \end{aligned} \tag{6.47}$$

The induction hypothesis shows that

$$-U_{j-1}\theta_{j-1} + \beta_{j-1}AU_{j-1}\theta_{j-1} - \sum_{i=1}^{j-1} s_{ij}u_i \in \mathcal{K}_j(A, f_0).$$

It remains to show that $f_{j-1} = b - Ax_{j-1} \in \mathcal{K}_j(A, f_0)$. For this, expand $b - Ax_{j-1}$ as

$$\begin{aligned} b - Ax_{j-1} &= b - Ax_{j-1} + Ax_{j-2} - Ax_{j-2} + \cdots - Ax_1 + Ax_0 - Ax_0 \\ &= \sum_{i=1}^{j-1} -A(x_i - x_{i-1}) + f_0. \end{aligned}$$

From the relation (6.47) applied with j replaced by i, we see that $x_i - x_{i-1}$ is a linear combination of $u_1, u_2, \ldots, u_i$, i.e. a member of $\mathcal{K}_i$, by the induction hypothesis. Therefore $-A(x_i - x_{i-1}) \in \mathcal{K}_{i+1}$, but since $i \leq j-1$ this vector belongs to $\mathcal{K}_j$. The remaining term f_0 is clearly in $\mathcal{K}_j$. Because $U_j = -A^{-1}Q_j$ has full column rank and $u_i \in \mathcal{K}_j(A, f_0)$ for $i = 1, \ldots, j$, U_j forms a basis of $\mathcal{K}_j(A, f_0)$. This completes the induction proof. □

From (6.29), we see that in the linear case under consideration, the vector $\bar{f}_j$ is the residual for $\bar{x}_j$:

$$\begin{aligned} \bar{f}_j = f_j - Q_j\theta_j &= (b - Ax_j) - Q_j\theta_j \\ &= (b - Ax_j) + AU_j\theta_j \\ &= b - A(x_j - U_j\theta_j) \\ &= b - A\bar{x}_j. \end{aligned} \tag{6.48}$$

The next theorem shows that $\bar{x}_j$ minimizes $\|b - Ax\|_2$ over the affine space $x_0 + \mathcal{K}_j(A, f_0)$.

Theorem 6.1. The vector $\bar{x}_j$ generated at the jth step of AA-TGS(∞) minimizes the residual norm $\|b - Ax\|_2$ over all vectors x in the affine space $x_0 + \mathcal{K}_j(A, f_0)$. It also minimizes the same residual norm over the subspace $x_k + \mathcal{K}_j(A, f_0)$ for any k such that $0 \leq k \leq j$.

Proof. Consider a vector of the form $x = x_j - \delta$, where $\delta = U_j y$ is an arbitrary member of $\mathcal{K}_j(A, f_0)$. We have

$$b - Ax = b - A(x_j - U_j y) = f_j + AU_j y = f_j - Q_j y. \tag{6.49}$$

The minimal norm $\|b - Ax\|_2$ is reached when $y = Q_j^\top f_j$ and the corresponding optimal x is $\bar{x}_j$. Therefore $\bar{x}_j$ is the vector x of the affine space $x_j + \mathcal{K}_j(A, f_0)$ with the smallest residual norm. We now write x as

$$\begin{aligned} x &= x_j - U_j y \\ &= x_0 + (x_1 - x_0) + (x_2 - x_1) + (x_3 - x_2) + \cdots + (x_{i+1} - x_i) \\ &\quad + \cdots + (x_j - x_{j-1}) - U_j y \qquad (6.50) \\ &= x_0 + \Delta x_0 + \Delta x_1 + \cdots + \Delta x_{j-1} - U_j y. \qquad (6.51) \end{aligned}$$

We now exploit the relation obtained from the QR factorization of Algorithm 5, namely $\mathcal{X}_j = U_j S_j$ in (6.30): if e is the vector of all ones, then

$$\Delta x_0 + \Delta x_1 + \cdots + \Delta x_{j-1} = \mathcal{X}_j e = U_j S_j e.$$

Define $t_j \equiv S_j e$. Then from (6.51) we obtain

$$x = x_j - \delta = x_0 - U_j[y - t_j]. \tag{6.52}$$

Hence the set of all vectors of the form $x_j - \delta = x_j - U_j y$ is the same as the set of all vectors of the form $x_0 - \delta'$, where $\delta' \in \mathcal{K}_j(A, f_0)$. As a result, $\bar{x}_j$ also minimizes $b - Ax$ over all vectors in the affine space $x_0 + \mathcal{K}_j(A, f_0)$.

The proof can be easily repeated if we replace x_0 with x_k for any k between 0 and j. The expansion (6.50)–(6.51) becomes

$$\begin{aligned} x_k - U_j y &= x_k + (x_{k+1} - x_k) + (x_{k+2} - x_{k+1}) \\ &\quad + \cdots + (x_{i+1} - x_i) + \cdots + (x_j - x_{j-1}) - U_j y \qquad (6.53) \\ &= x_k + \Delta x_k + \Delta x_{k+1} + \cdots + \Delta x_{j-1} - U_j y. \qquad (6.54) \end{aligned}$$

The rest of the proof is similar and straightforward. □

Theorem 6.1 shows that $\bar{x}_j$ is the jth iterate of the GMRES algorithm for solving $Ax = b$ with the initial guess x_0 and that $\bar{f}_j$ is the corresponding residual. The value of $\bar{x}_j$ is independent of the choice of β_i for $i \le j$. Now consider the residual f_{j+1} of AA-TGS(∞) at step $j + 1$. From the relations $x_{j+1} = \bar{x}_j + \beta_j \bar{f}_j$ and (6.48), we get

$$f_{j+1} = b - A[\bar{x}_j + \beta_j \bar{f}_j] = b - A\bar{x}_j - \beta_j A\bar{f}_j = \bar{f}_j - \beta_j A\bar{f}_j = (I - \beta_j A)\bar{f}_j. \tag{6.55}$$

This implies that the vector f_{j+1} is the residual for x_{j+1} obtained from $x_{j+1} = \bar{x}_j + \beta_j \bar{f}_j$, which is a simple Richardson iteration starting from the iterate $\bar{x}_j$. Therefore x_{j+1} in line 15 of Algorithm 5 is simply a Richardson iteration step from this GMRES iterate. This is stated in the following proposition.

Proposition 6.2. The residual f_{j+1} of the iterate x_{j+1} generated at the jth step of AA-TGS(∞) is equal to $(I - \beta_j A)\bar{f}_j$, where $\bar{f}_j = b - A\bar{x}_j$ minimizes the residual norm $\|b - Ax\|_2$ over all vectors x in the affine space $x_0 + \mathcal{K}_j(A, f_0)$. In other words, the $(j + 1)$th iterate of AA-TGS(∞) can be obtained by performing one step of a Richardson iteration applied to the jth GMRES iterate.

A similar result has also been proved for the standard AA by Walker and Ni (2011) under slightly different assumptions; see Section 6.7.5.

Convergence in the linear case can therefore be analysed by relating the residual of full AA-TGS to that of GMRES. The following corollary of the above proposition shows a simple but useful inequality.

Corollary 1. If AA-TGS(∞) is used to solve the system (3.1), then the residual norm of the iterate $x_{j+1}^{(AA\text{-}TGS)}$ satisfies the inequality

$$\|b - Ax_{j+1}^{(AA\text{-}TGS)}\|_2 \le \|(I - \beta A)\|_2 \, \|b - Ax_j^{(GMRES)}\|_2, \tag{6.56}$$

where $x_j^{(GMRES)}\|_2$ is the iterate obtained by j steps of full GMRES starting with the same initial guess x_0.

Proof. According to Proposition 6.2, $r_{j+1} = -f(x_{j+1}) = -(I-\beta A)\bar{f}_j = (I-\beta A)r_j$, where r_j is the residual obtained from j steps of GMRES starting with the same initial guess x_0. Therefore

$$\begin{aligned}\left\|b - Ax_{j+1}^{(AA\text{-}TGS)}\right\|_2 &= \left\|(I-\beta A)\left(b - Ax_j^{(GMRES)}\right)\right\|_2 \\ &= \left\|(I-\beta A)r_j^{(GMRES)}\right\|_2 \\ &\le \|(I-\beta A)\|_2 \left\|r_j^{(GMRES)}\right\|_2. \end{aligned} \tag{6.57}$$

□

Thus essentially all the convergence analysis of GMRES can be adapted to AA-TGS when it is applied to linear systems. The next section examines the special case of linear symmetric systems.

6.7.4. The linear symmetric case

A simple experiment will reveal a remarkable observation for the linear case when the matrix A is symmetric. Indeed, the orthogonalization process (lines 6–10 of Algorithm 5) simplifies in this case in the sense that S_j consists of only three non-zero diagonals in the upper triangular part when A is symmetric. In other words, when A is symmetric we only need to save q_{j-2}, q_{j-1} and u_{j-2}, u_{j-1} in order to generate q_j and u_j in the full-depth case, i.e. when $m = \infty$. This is similar to the simplification obtained by a Krylov method like FOM or GMRES when the matrix is symmetric. We first examine the components of the vector $Q_j^\top f_j$ in line 14 of Algorithm 5.

Lemma 6.2. When $f(x) = b - Ax$, where A is a real non-singular symmetric matrix, then the entries of the vector $\theta_j = Q_j^\top f_j$ in Algorithm 5 are all zeros except the last two.

Proof. Let $i \le j-1$. From (6.55), we have

$$(f_j, q_i) = (\bar{f}_{j-1} - \beta_{j-1}A\bar{f}_{j-1}, q_i) = (\bar{f}_{j-1}, q_i) - \beta_{j-1}(A\bar{f}_{j-1}, q_i).$$

The first term on the right-hand side vanishes because

$$(\bar{f}_{j-1}, q_i) = (f_{j-1} - Q_{j-1}\theta_{j-1}, q_i) = ((I - Q_{j-1}Q_{j-1}^\top)f_{j-1}, q_i) = 0.$$

For the second term we write $(A\bar{f}_{j-1}, q_i) = (\bar{f}_{j-1}, Aq_i)$, and observe that since $u_i \in \mathcal{K}_i(A, f_0)$ then $q_i = -Au_i$ belongs to the Krylov subspace $\mathcal{K}_{i+1}(A, f_0)$, which is the same as $\mathrm{Span}\{U_{i+1}\}$ according to Lemma 6.1. Thus it can be written as $q_i = -Au_i = U_{i+1}y$ for some y and hence $Aq_i = AU_{i+1}y = -Q_{i+1}y$, that is, Aq_i is in the span of $q_1, \ldots, q_{i+1}$. Therefore, recalling that $\bar{f}_{j-1} \perp \mathrm{Span}\{Q_{j-1}\}$, we have

$$(\bar{f}_{j-1}, Aq_i) = 0 \quad \text{for } i \le j-2. \tag{6.58}$$

In the end, we obtain $(f_j, q_i) = 0$ for $i \le j-2$. □

Lemma 6.2 indicates that the computation of x_{j+1} in line 15 of Algorithm 5 only depends on the two most recent q_i and u_i. In addition, as is shown next, the vectors q_j and u_j in line 12 can be computed from q_{j-2}, q_{j-1} and u_{j-2}, u_{j-1} instead of all previous q_i and u_i.

Theorem 6.2. When $f(x) = b - Ax$, where A is a real non-singular symmetric matrix, then the upper triangular matrix S_j produced in Algorithm 5 is banded with bandwidth 3, that is, we have $s_{ik} = 0$ for $i < k - 2$.

Proof. It is notationally more convenient to consider column $k + 1$ of S_j, where $k + 1 \leq j$. Let $\Delta f_k \equiv f_{k+1} - f_k$ and $\Delta x_k = x_{k+1} - x_k$. The inner product $s_{i,k+1}$ in Algorithm 5 is the same as $s_{i,k+1} = (\Delta f_k, q_i)$ that would be obtained by a classical Gram–Schmidt algorithm. We note that for $i \leq k$ we have $s_{i,k+1} = -(A\Delta x_k, q_i)$. Exploiting the relation

$$\Delta x_k = (\bar{x}_k + \beta_k \bar{f}_k) - x_k = x_k - U_k\theta_k + \beta_k \bar{f}_k - x_k = -U_k\theta_k + \beta_k \bar{f}_k,$$

we write

$$\begin{aligned} A\Delta x_k &= -AU_k\theta_k + \beta_k A\bar{f}_k \\ &= Q_k\theta_k + \beta_k A\bar{f}_k \\ &= -(f_k - Q_k\theta_k) + f_k + \beta_k A\bar{f}_k \\ &= -\bar{f}_k + f_k + \beta_k A\bar{f}_k, \end{aligned}$$

and hence

$$(A\Delta x_k, q_i) = -(\bar{f}_k, q_i) + (f_k, q_i) + \beta_k (A\bar{f}_k, q_i). \tag{6.59}$$

The first term on the right-hand side, $(\bar{f}_k, q_i)$, vanishes since $i \leq k$. According to Lemma 6.2 the inner product (f_k, q_i) is zero for $i \leq k - 2$. In the proof of Lemma 6.2 we showed that $(\bar{f}_{j-1}, Aq_i) = 0$ for $i \leq j - 2$ (see (6.58)), which means that $(\bar{f}_k, Aq_i) = 0$ for $i \leq k - 1$. In the end $s_{i,k+1} = -(A\Delta x_k, q_i) = 0$ for $i < k - 1$, which is equivalent to the desired result. □

Lemma 6.2 and Theorem 6.2 show that when AA-TGS(∞) is applied to solving linear symmetric problems, only the two most recent q_i and u_i, i.e. q_{j-2}, q_{j-1} and u_{j-2}, u_{j-1}, are needed to compute the next iterate x_{j+1}. In other words, the **for** loop in line 6 of the algorithm only needs to be executed for $i = j-2$ and $i = j-1$, which means that AA-TGS(3) is equivalent to AA-TGS(∞) in the linear symmetric case. In practice, this leads to a significant reduction in memory and computational cost.

We saw earlier that in all cases the full AA-TGS algorithm is equivalent to the full window Anderson, at least in exact arithmetic. AA-TGS is just a different implementation of AA in this case. In the linear case, Proposition 6.2 states that

the full AA-TGS is equivalent to (full) GMRES followed by a Richardson step. This led to Corollary 1, which enables us to establish convergence results by exploiting already known theory. One specific such result is an analysis of the special case when the problem is linear and symmetric.

Theorem 6.3. Assume that A is symmetric positive definite and that a constant β is used in AA-TGS. Then the iterate $x_{j+1}^{(AA\text{-}TGS)}$ obtained at the $(j+1)$th step of AA-TGS(∞) satisfies

$$\begin{aligned}\|b - Ax_{j+1}^{(AA\text{-}TGS)}\|_2 &\le \|I - \beta A\|_2 \frac{\|b - Ax_0\|_2}{T_j\left(\frac{\kappa+1}{\kappa-1}\right)} \\ &\le 2\|I - \beta A\|_2 \|b - Ax_0\|_2 \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^j,\end{aligned}$$

where T_j is the Chebyshev polynomial of the first kind of degree j, and $\kappa = \kappa(A)$ is the 2-norm condition number of A.

Proof. We start from inequality (6.56) of Corollary 1. An analysis similar to that in Saad (2003, § 6.11.3) for the CG method will show that

$$\|r_j^{GMRES}\|_2 \le \frac{\|r_0\|_2}{T_j(1+2\eta)},$$

in which $\eta = \lambda_{\min}/(\lambda_{\max} - \lambda_{\min}) = 1/(\kappa - 1)$. Noting that $1 + 2\eta = (\kappa + 1)/(\kappa - 1)$ establishes the first inequality. The second one follows from using standard expressions of the Chebyshev polynomials based on the hyperbolic cosine (Saad 2003, pp. 204–205), which shows that

$$T_j(1+2\eta) \ge \frac{1}{2}\left(\frac{\sqrt{\kappa(A)}+1}{\sqrt{\kappa(A)}-1}\right)^j.$$

This completes the proof. □

6.7.5. Other links between AA and Krylov methods in the linear case

The analysis shown above establishes strong connections between full-depth AA-TGS and GMRES. Since AA-TGS is equivalent to standard AA in the full window case, these results are also valid for AA. Such connections were established well before the recent article by Tang *et al.* (2024).

Specifically, Walker and Ni (2011) studied the algorithm and showed a form of equivalence between AA and GMRES in the linear case. Another study along the same lines, discussed at the end of this section, is that of Haelterman, Degroote, Heule and Vierendeels (2010), which is concerned with a slightly different version of AA.

Because the Walker and Ni result is somewhat different from that of Proposition 6.2 seen in Section 6.7.3, we will now summarize it. Their paper (Walker and Ni 2011) makes the following set of assumptions.

Assumption (A).

- AA is applied to the fixed-point mapping $g(x) = Ax + b$.
- Anderson acceleration is not truncated, i.e. $m = \infty$.
- $(I - A)$ is non-singular.
- GMRES is used to solve $(I - A)x = b$ with the same initial guess x_0 as for AA.

The main result of the article is stated for the formulation of AA that follows the notation of Pulay mixing seen in Section 4.4. Accounting for this change of notation, their result is stated below.

Theorem 6.4. Suppose that Assumption (A) holds and that, for some $j > 0$,

$$r_{j-1}^{(GMRES)} \neq 0,$$

and also that

$$\left\|r_{i-1}^{(GMRES)}\right\|_2 > \left\|r_i^{(GMRES)}\right\|_2$$

for $0 < i < j$. Then

$$\bar{x}_j = x_j^{(GMRES)} \quad \text{and} \quad x_{j+1} = g\left(x_j^{(GMRES)}\right).$$

Degroote, Bathe and Vierendeels (2009) described a method called QN-ILS (quasi-Newton inverse least-squares), which resembles AA although it is presented as a quasi-Newton approach (hence its name). They seemed unaware of the Anderson article and the related literature but the spirit of their method is quite close to that of AA. In fact, for their method they even use the same formulation as that of the original AA, in that they invoke the basis (6.1) instead of the bases using forward differences. Their algorithm is viewed from the angle of a quasi-Newton method with updates of type II, where the inverse Jacobian is approximated. Using our notation, the only difference with AA is that in their case the update (6.9) for the new iterate becomes

$$x_{j+1} = \bar{x}_j + f_j. \tag{6.60}$$

Thus, relative to the update equation (6.9) of AA, β is set to one and $\bar{f}_j$ is replaced by f_j in QN-ILS. In the linear case, and the full window case, the two methods are mathematically equivalent since $\bar{f}_j = f_j - \mathcal{F}_j\gamma^{(j)}$, and the two methods will produce the same space of approximants in the projection process, at each step. In the nonlinear case, the two methods will not generate the same iterates in general.

From an implementation perspective, QN-ILS is more expensive than AA. As described in Degroote *et al.* (2009), the algorithm recomputes a new QR factorization each time, and does not exploit any form of downdating. The main reason for this is that, as already mentioned, QN-ILS relies on a basis of the form (6.1), which changes entirely at each new step j. This also means that each of the vectors $d_i = f_j - f_i$ for $i = ([j - m]), \ldots, j - 1$ and the related differences $x_j - x_i$ must be

recomputed at every step j, leading to a substantial added cost when compared to modern implementations of AA. Indeed, the basis of the Δx_i used in the modern version of AA requires only that we compute the most recent pair $\Delta f_{j-1}, \Delta x_{j-1}$, since the other needed pairs were computed in earlier steps. In AA, one column is computed and added to $\mathcal{F}_j$, and one is dropped from it (when $j > m$). Similarly for $\mathcal{X}_j$.

Haelterman *et al.* (2010) studied the method in the linear case, and established that it is equivalent to GMRES in this situation. This result is similar to that of Walker and Ni (2011), but we need to remember that AA and QN-ILS are different in the nonlinear case.

6.7.6. Convergence properties of AA

Toth and Kelley (2015) proved that AA is locally r-linearly convergent under the condition that the fixed-point map g is a contraction mapping and the coefficients in the linear combination remain bounded. A number of other results were proved under different assumptions.

The article by Toth and Kelley (2015) starts by considering the linear case in which $g(x) = Mx + b$, and shows that when M is contracting with $\|M\| = c < 1$ then the iterates of Anderson acceleration applied to g will converge to the fixed point $x^* = (I - M)^{-1}b$. In addition, the residuals converge q-linearly to zero, that is, if $f(x) = g(x) - x$ then $\|f(x_{k+1})\| \leq c\|f(x_k)\|$. This is used as a starting point for proving a result in the nonlinear case.

Consider the situation where AA is used to find the fixed point of a function g and let $f(x) = g(x) - x$. Toth and Kelley (2015) invoke formulation (4.15)–(4.17) because their results require assumptions on the coefficients θ_i. With this in mind, their main result can be stated as follows.

Theorem 6.5 (Toth and Kelley 2015, Theorem 2.3). Assume the following.

(1) There is a constant μ_θ such that $\sum_{i=j-j_m}^{j} |\theta_i| \leq \mu_\theta$ for all $j \geq 0$.

(2) There is an x^* such that $f(x^*) = g(x^*) - x^* = 0$.

(3) g is Lipschitz-continuously differentiable in the ball $\mathcal{B}(\hat{\rho}) = \{x \mid \|x - x^*\| \leq \hat{\rho}\}$.

(4) There is a $c \in (0, 1)$ such that $\|g(u) - g(v)\| \leq c\|u - v\|$ for all u, v in $\mathcal{B}(\hat{\rho})$.

Let $c < \hat{c} < 1$. Then, if x_0 is sufficiently close to x^*, the Anderson iteration converges to x^* r-linearly with r-factor no greater than $\hat{c}$. Specifically,

$$f(x_k) \leq \hat{c}^k f(x_0), \tag{6.61}$$

and

$$\|x_k - u^*\| \leq \frac{1+c}{1-c}\hat{c}^k \|x_0 - u^*\|. \tag{6.62}$$

Not that the result is valid for any norm, not just for the case when the 2-norm minimization is used in (4.13). The first condition only states that the coefficients θ_i

resulting from the constrained least-squares problem (4.17) (or equivalently the unconstrained problem (4.13)) all remain bounded in magnitude. It cannot be proved that this condition will be satisfied and the ill-conditioning of the least-squares problem may lead to large values the θ_i. However, Toth and Kelley (2015) show how to modify the standard AA scheme to enforce the boundedness of the coefficients in practice.

In addition, Toth and Kelley (2015) consider the particular case when the window size is $m = 1$ and show that in this situation the coefficients θ_i are bounded if c is sufficiently small that $\hat{c} \equiv ((3c - c^2)/(1 - c)) < 1$. If this condition is satisfied and if $x_0 \in \mathcal{B}(\hat{\rho})$, then they show that AA(1) with least-squares optimization converges q-linearly with q-factor $\hat{c}$.

Even though these results are proved under somewhat restrictive assumptions, they nevertheless establish strong theoretical convergence properties. In particular, the results show that under certain conditions the AA-accelerated iterates will converge to the solution at least as fast as the original fixed-point sequence.

The theory in the Toth–Kelley article does not prove that the convergence of an AA accelerated sequence will be faster than that of the original fixed-point iteration. Evans, Pollock, Rebholz and Xiao (2020) address this issue by showing theoretically that Anderson acceleration (AA) does improve the convergence rate of contractive fixed-point iterations in the vicinity of the fixed point. Their experiments illustrate the improved linear convergence rates. However, they also show that when the initial fixed-point iteration converges quadratically, then its convergence is slowed by the AA scheme.

Pollock and Rebholz (2021) discuss further theoretical aspects of the AA algorithm and show a number of strategies to improve convergence. These include techniques for adapting the window size m dynamically, as well as filtering out columns of $\mathcal{F}_j$ when linear dependence is detected. Along the same lines, building on work by Rohwedder (2010), Brezinski, Cipolla, Redivo-Zaglia and Saad (2022) present a *stabilized version of AA* which examines the linear independence of the latest Δf_j from previous differences. The main idea is to ensure that we keep a subset of the differences that are *sufficiently linearly independent* for the projection process needed to solve the least-squares problem. Local convergence properties are proved under some assumptions.

It has been observed that AA works fairly well in practice, especially in the situation when the underlying fixed-point iteration that is accelerated has adequate convergence properties. However, without any modifications, it is not possible to guarantee that the method will converge. A few papers address this 'global convergence' issue. Zhang, O'Donoghue and Boyd (2020) consider 'safeguarding strategies' to ensure global convergence of type I AA methods. Their technique assumes that the underlying fixed-point mapping g is non-expansive and, adopting a multisecant viewpoint, develop a type I based AA update whereby the focus is to approximate the Jacobian instead of its inverse as is done in AA. Their main scheme relies on two ingredients. The first is to add a regularization of the approximate

Jacobian to deal with the potential (near-) singularity of the approximate Jacobian. The second is to interleave the AA scheme with a linear mixing scheme of the form (4.9). This is done in order to 'safeguard the decrease in the residual norms'.

7. Nonlinear truncated GCR

Krylov accelerators for linear systems, which were reviewed in Section 3.3, can be adapted in a number of ways for nonlinear problems. We already noted that AA can be viewed as a modified Krylov subspace method in the linear case. We also showed strong links between Krylov methods and a few extrapolation techniques in Section 2. One way to uncover generalizations of Krylov methods for nonlinear equations is to take a multisecant viewpoint. The process begins with a subspace spanned by a set of vectors $\{p_{[j-m+1]}, p_{[j-m+1]+1}, \ldots, p_j\}$ – typically related to a Krylov subspace – and finds an approximation to the Jacobian or its inverse when it is restricted to this subspace. This second step can take different forms, but it is typically expressed as a multisecant requirement, whereby a set of vectors v_i are coupled to the vectors p_i such that

$$v_i \approx J(x_i)p_i, \tag{7.1}$$

where $J(x_i)$ is the Jacobian of f at x_i. Observe that a different Jacobian is involved for each index i. There are a number of variations to this scheme. For example, $J(x_i)$ can be replaced by a fixed Jacobian at some other point, e.g. $x_{[j-m+1]}$ or x_0 as in inexact Newton methods.

We will say that the two sets

$$P_j = [p_{[j-m+1]}, p_{[j-m+1]+1}, \ldots, p_j], \quad V_j = [v_{[j-m+1]}, v_{[j-m+1]+1}, \ldots, v_j] \tag{7.2}$$

are *paired.* This setting was encountered in the linear case (see (3.28)) and in Anderson acceleration, where P_j was just the set $\mathcal{X}_j$ and V_j was $\mathcal{F}_j$. Similarly, in AA-TGS these two sets were U_j and Q_j respectively. It is possible to develop *a broad class of multi-purpose accelerators with this general viewpoint.* One of these methods (He *et al.* 2024), named the nonlinear truncated generalized conjugate residual (nlTGCR), is built as a nonlinear extension of the GCR method seen in Section 3.3.3. It is discussed next.

7.1. nlTGCR

Recall that in the linear case, where we solve the system linear $Ax = b$, the main ingredient of GCR is to build two sets of paired vectors $\{p_i\}, \{Ap_i\}$, where the p_i are the search directions obtained in earlier steps and the Ap_i are orthogonal to each other. At the jth step we introduce a new pair p_{j+1}, Ap_{j+1} to the set in which p_{j+1} is initially set equal to the most recent residual; see lines 7–12 of Algorithm 3. This vector is then $A^\top A$-orthogonalized against the previous p_i. We saw that this process leads to a simple expression for the approximate solution, using a projection mechanism; see Lemma 3.1.

The next question we address is how to extend GCR, or its truncated version TGCR, to the nonlinear case. The simplest approach is to exploit an inexact Newton viewpoint in which the GCR algorithm is invoked to approximately solve the linear systems that arise from Newton's method. However, this is avoided for a number of reasons. First, unlike quasi-Newton techniques, inexact Newton methods build an approximate Jacobian for the current iterate, and this approximation is used only for the current step. In other words, it is discarded after it is used and another one is built in the next step. This is to be contrasted with quasi-Newton or multisecant approaches where these approximations are built gradually. Inexact Newton methods perform best when the Jacobian is explicitly available or can be inexpensively approximated. In such cases it is possible to solve the system in Newton's method as accurately as desired, leading to superlinear convergence.

An approach that is more appealing for fixed-point iterations is to exploit the multisecant viewpoint sketched above, by adapting it to GCR. At a given step j of TGCR, we would have available the previous directions $p_{[j-m+1]}, \ldots, p_j$ along with their corresponding (paired) v_i, for $i = [j-m+1], \ldots, j$. In the linear case, each v_i equals Ap_i. In the nonlinear case we would instead have $v_i \approx J(x_i)p_i$. The next pair p_{j+1}, v_{j+1} is obtained by the update

$$p_{j+1} = r_{j+1} - \sum_{i=[j-m+1]}^{j} \beta_{ij} p_i, \quad v_{j+1} = J(x_{j+1})r_{j+1} - \sum_{i=[j-m+1]}^{j} \beta_{ij} v_i, \tag{7.3}$$

where the β_{ij} are selected so as to make v_{j+1} orthonormal to the previous vectors $v_{[j-m+1]}, \ldots, v_j$. One big difference from the linear case is that the residual vector r_{j+1} is now the nonlinear residual, which is $r_{j+1} = -f(x_{j+1})$.

This process builds a pair (p_{j+1}, v_{j+1}) such that v_{j+1} is orthonormal to the previous vectors $v_{[j-m+1]}, \ldots, v_j$. The current 'search' directions $\{p_i\}$ *for* $i = [j-m+1], \ldots, j$ are *paired* with the vectors $v_i \approx J(x_i)p_i$, *for* $i = [j-m+1], \ldots, j$; see (7.2).

Another important difference from TGCR is that the way in which the solution is updated in line 6 of Algorithm 3 is no longer valid. This is because the second part of Lemma 3.1 no longer holds in the nonlinear case. Therefore the update will be of the form $x_j + P_j y_j$ where $y_j = V_j^\top r_j$. Putting these together leads to the nonlinear adaptation of GCR shown in Algorithm 6.

The relation to Newton's method can be understood from the local linear model which is at the foundation of the algorithm

$$f(x_j + P_j y) \approx f(x_j) + V_j y, \tag{7.4}$$

which follows from the following approximation, where the γ_i are the components of y and the sum is over $i = [j-m+1]$ to j:

$$f(x_j + P_j y) \approx f(x_j) + \sum \gamma_i J(x_j)p_i \approx f(x_j) + \sum \gamma_i J(x_i)p_i \approx f(x_j) + V_j y.$$

Algorithm 6 nlTGCR(m)

1: **Input:** Function $f(x)$, initial guess x_0, window size m
2: Set $r_0 = -f(x_0)$.
3: Compute $v = J(x_0)r_0$; ▷ *Use Fréchet*
4: $v_0 = v/\|v\|_2$, $p_0 = r_0/\|v\|_2$;
5: **for** $j = 0, 1, 2, \ldots,$ **do**
6: $y_j = V_j^\top r_j$
7: $x_{j+1} = x_j + P_j y_j$ ▷ *Scalar α_j becomes vector y_j*
8: $r_{j+1} = -f(x_{j+1})$ ▷ *Replaces linear update: $r_{j+1} = r_j - V_j y_j$*
9: Set: $p := r_{j+1}$; and compute $v = J(x_{j+1})p$ ▷ *Use Fréchet*
10: Compute $\beta_j = V_j^\top v$
11: $v = v - V_j\beta_j, \quad p = p - P_j\beta_j$
12: $p_{j+1} := p/\|v\|_2$; $\quad v_{j+1} := v/\|v\|_2$;
13: **end for**

The method essentially minimizes the residual norm of the linear model (7.4) at the current step. Recall that Anderson exploited a similar local relation represented by (6.10), and the intermediate solution $\bar{x}_k$ in (6.7) is a local minimizer of the linear model. We will often use the notation

$$V_j \approx [J]P_j \tag{7.5}$$

to express the relation represented by equation (7.1).

7.2. *Linear updates*

The reader may have noticed that Algorithm 6 requires two function evaluations per step, one in line 8 where the residual is computed, and one in line 9 when invoking the Fréchet derivative to compute $J(x_{j+1})p$ using the formula

$$J(x)p \approx \frac{f(x + \epsilon p) - f(x)}{\epsilon}, \tag{7.6}$$

where ϵ is some carefully selected small scalar. It is possible to avoid calculating the nonlinear residual by simply replacing it with its linear approximation given by expression (7.4), from which we get

$$r_{j+1} = -f(x_j + P_j y_j) \approx -f(x_j) - V_j y_j = r_j - V_j y_j.$$

Therefore the idea of this 'linearized update version' of nlTGCR is to replace r_{j+1} in line 8 with its linear approximation $r_j - V_j y_j$:

$$8a: \quad r_{j+1} = r_j - V_j y_j.$$

This is now a method that resembles an inexact Newton approach. It will be equivalent to it if we add one more modification to the scheme, namely that we omit

updating the Jacobian in line 9 when computing v. In other words, the Jacobian $J(x_{j+1})$ invoked in line 9 is constant and equal to $J(x_0)$, and line 9 becomes

$$9a: \quad \text{Set: } p := r_{j+1}; \text{ and compute } v = J(x_0)p.$$

In practice this means that when computing the vector v in line 9 of Algorithm 6 with equation (7.6), the vector x is set to x_0. This works with restarts, that is, when the number of steps reaches a restart dimension, or when the linear residual has shown sufficient decrease, x_0 is reset to be the latest iteration computed, and a new subspace and corresponding approximation are computed.

All this means is that with minor changes to Algorithm 6 we can implement a whole class of methods that have been thoroughly studied in the past; see e.g. Dembo *et al.* (1982), Brown and Saad (1990, 1994) and Eisenstat and Walker (1994), among others. Probably the most significant disadvantage of inexact Newton methods, or to be specific Krylov–Newton methods, is that a large number of function evaluations may be needed to build the Krylov subspace in order to obtain a single iterate, i.e. the next (inexact) Newton iterate. After this iterate is computed, all the information gathered at this step, specifically P_k and V_k, is discarded. This is to be contrasted with quasi-Newton techniques, where the most recent function evaluation contributes to building an updated approximate Jacobian. Inexact Newton methods are most successful when the Jacobian is available, or inexpensive to compute, and some effective preconditioner can be readily computed.

Nevertheless, it may still be cost-effective to reduce the number of function evaluations from two to one whenever possible. We can update the residual norm by replacing line 8 of Algorithm 6 with the linear form 8a when it is deemed safe, that is, typically after the iteration reaches a region where the iterate is close enough to the exact solution that the linear model (7.4) is accurate enough. A simple strategy to employ linear updates and move back to using nonlinear residuals was implemented and tested in He *et al.* (2024). It is based on probing periodically how far the linear residual $\tilde{r}_{j+1} = r_j - V_j y_j$ is from the actual one. Define

$$d_j = 1 - \frac{(\tilde{r}_j, r_j)}{\|\tilde{r}_j\|_2 \|r_j\|_2}. \tag{7.7}$$

The *adaptive* nlTGCR switches the linear mode on when $d_j < \tau$ and returns to the nonlinear mode if $d_j \geq \tau$, where τ is a small threshold parameter. In the experiments discussed in Section 7.5, we set $\tau = 0.01$.

7.3. *Nonlinear updates*

We now consider an implementation of Algorithm 6 in which nonlinear residuals are computed at each step. We can study the algorithm by establishing relations with the linear residual

$$\tilde{r}_{j+1} = r_j - V_j y_j, \tag{7.8}$$

and the deviation between the actual residual r_{j+1} and its linear version $\tilde{r}_{j+1}$ at the $(j+1)$th iteration:

$$z_{j+1} = \tilde{r}_{j+1} - r_{j+1}. \tag{7.9}$$

To analyse the magnitude of z_{j+1}, we define

$$w_i = (J(x_j) - J(x_i))p_i \text{ for } i = [j-m+1], \ldots, j, \quad W_j = [w_{[j-m+1]}, \ldots, w_j] \tag{7.10}$$

and

$$s_j = f(x_{j+1}) - f(x_j) - J(x_j)(x_{j+1} - x_j). \tag{7.11}$$

Observe that

$$J(x_j)p_i = J(x_i)p_i + w_i = v_i + w_i. \tag{7.12}$$

Recall from the Taylor series expansion that s_j is a second-order term relative to $\|x_{j+1} - x_j\|_2$. Then it can be shown (He *et al.* 2024) that the difference $\tilde{r}_{j+1} - r_{j+1}$ satisfies the relation

$$\tilde{r}_{j+1} - r_{j+1} = W_j y_j + s_j = W_j V_j^\top r_j + s_j, \tag{7.13}$$

and therefore that

$$\|\tilde{r}_{j+1} - r_{j+1}\|_2 \le \|W_j\|_2 \, \|r_j\|_2 + \|s_j\|_2. \tag{7.14}$$

When the process nears convergence, $\|W_j\|_2 \|r_j\|_2$ is the product of two first-order terms while s_j is a second-order term according to its definition (7.11). Thus z_{j+1} is a quantity of the second order.

The following properties of Algorithm 6 are easy to establish; see He *et al.* (2024) for the proof and other details. We let m_j denote the number of columns in V_j and P_j, i.e. $m_j = \min\{m, j+1\}$.

Proposition 7.1. The following properties are satisfied by the vectors produced by Algorithm 6.

(1) $(\tilde{r}_{j+1}, v_i) = 0$ for $[j-m+1] \le i \le j$, i.e. $V_j^\top \tilde{r}_{j+1} = 0$.

(2) $\|\tilde{r}_{j+1}\|_2 = \min_y \| - f(x_j) + [J]P_j y\|_2 = \min_y \| - f(x_j) + V_j y\|_2$.

(3) $\langle v_{j+1}, \tilde{r}_{j+1} \rangle = \langle v_{j+1}, r_j \rangle$.

(4) $y_j = V_j^\top r_j = \langle v_j, \tilde{r}_j \rangle e_{m_j} - V_j^\top z_j$ where $e_{m_j} = [0, 0, \ldots, 1]^\top \in \mathbb{R}^{m_j}$.

Property (4) and equation (7.14) suggest that when z_j is small, then y_j will have small components everywhere except for the last component. This happens when the model is *close to being linear* or when it is nearing convergence.

7.4. *Connections with multisecant methods*

The update at step j of nlTGCR can be written as

$$x_{j+1} = x_j + P_j V_j^\top r_j = x_j + P_j V_j^\top (-f(x_j)),$$

showing that nlTGCR is a multisecant-type method in which the inverse Jacobian at step j is approximated by

$$G_{j+1} \equiv P_j V_j^\top. \tag{7.15}$$

This approximation satisfies the *multisecant* equation

$$G_{j+1} v_i = p_i \quad \text{for } [j-m+1] \le i \le j. \tag{7.16}$$

Indeed, $G_{j+1} v_i = P_j V_j^\top v_i = p_i = J(x_i)^{-1} v_i$ for $[j-m+1] \le i \le j$. In other words G_{j+1} inverts $J(x_i)$ exactly when applied to v_i.

If we substitute p_i with Δx_j and v_i with Δf_j we see that equation (7.16) is just the constraint (5.10) we encountered in Broyden's second update method. In addition, the update G_{j+1} also satisfies the 'no-change' condition

$$G_{j+1} q = 0 \quad \text{for all } q \perp v_i, \quad \text{for } [j-m+1] \le i \le j. \tag{7.17}$$

The usual no-change condition for secant methods is of the form

$$(G_{j+1} - G_{[j-m+1]}) q = 0 \quad \text{for } q \perp \Delta f_i,$$

which in our case would become $(G_{j+1} - G_{[j-m+1]}) q = 0$ for $q \perp v_i$ for $[j-m+1] \le i \le j$. This means that we are in effect updating $G_{[j-m+1]} \equiv 0$. Interestingly, G_{j+1} satisfies an optimality result that is similar to that of other secant-type and multisecant-type methods. This is stated in the following proposition, which is easy to prove; see He *et al.* (2024).

Proposition 7.2. The unique minimizer of the optimization problem

$$\min\{\|G\|_F, G \in \mathbb{R}^{n \times n} \text{ subject to } G V_j = P_j\} \tag{7.18}$$

is the matrix $G_{j+1} = P_j V_j^\top$.

The condition $G_{j+1} V_j = P_j$ is the same as the multisecant condition $G_j \mathcal{F}_j = \mathcal{X}_j$ of equation (6.40) discussed when we characterized AA as a multisecant method. In addition, recall that the multisecant version of the no-change condition, as represented by equation (6.41), is satisfied. This is the same as the no-change condition seen above for nlTGCR.

Therefore we see that the two methods are quite similar, in that they are both multisecant methods defined from a pairing of two sets of vectors: V_j, P_j for nlTGCR on the one hand, and $\mathcal{F}_j$, $\mathcal{X}_j$ for AA on the other. From this viewpoint, the two methods differ mainly in the way in which the sets $\mathcal{F}_j / V_j$ and $\mathcal{X}_j / P_j$ are defined. Let us use the more general notation V_j, P_j for both pairs of subspaces.

In both cases, a vector v_j is related to the corresponding p_j by the fact that

$$v_j \approx J(x_j) p_j. \tag{7.19}$$

Looking at line 9 of Algorithm 6 indicates that before the orthogonalization step in nlTGCR (lines 10–12), this relation becomes an equality, or aims to be close to an equality, by employing a Fréchet differentiation. Thus the process introduces a

pair p_j, v_j to the current paired subspaces where p_j is accurately mapped to v_j by $J(x_j)$; see (7.19). In the case of AA we have $v_j = \Delta f_{j-1} = f_j - f_{j-1}$, and write

$$f_j \approx f_{j-1} + J(x_{j-1})(x_j - x_{j-1}) \rightarrow \Delta f_{j-1} \approx J(x_{j-1})\Delta x_{j-1}, \tag{7.20}$$

which is an expression of the form (7.19) for index $j - 1$.

An advantage of nlTGCR is that relation (7.19) is a *more accurate* representation of the Jacobian than relation (7.20), which can be a rough approximation when x_j and x_{j-1} are not close to each other. This advantage comes at the extra cost of an additional function evaluation, but this can be mitigated by an adaptive scheme, as was seen at the end of Section 7.2.

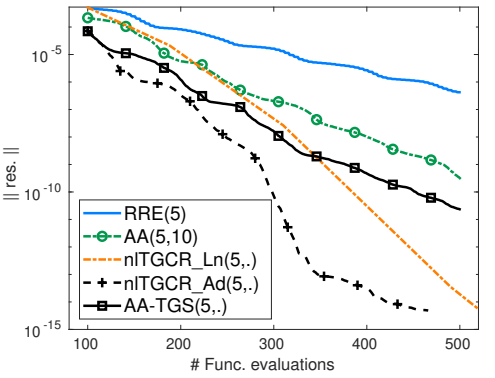
7.5. Numerical illustration: nlTGCR and Anderson

We now return to the numerical examples seen in Section 6.6 to test nlTGCR along with Anderson acceleration. As mentioned earlier, we can run nlTGCR in different modes. We can adopt a 'linear' mode, which is simply an inexact Newton approach where the Jacobian systems are solved with a truncated GCR method. It is also possible to run an 'adaptive' algorithm, as described earlier, where we switch between the linear and nonlinear residual modes with the help of a simple criterion; see the end of Section 7.2. Figure 7.1 reproduces the curves of `RRE(5)` and `AA(5,10)`, and `AA-TGS(5,.)` shown in Section 6.6, and adds results with `nlTGCR_Ln(5,.)`, `nlTGCR_Ad(5,.)`, the linear and adaptive versions of nlTGCR respectively. As before, the parameter 5 for these two runs represents the window size. What is shown is similar to what we saw in Section 6.6, and the parameters, such as residual tolerance and maximum number of iterations, are identical. One difference is that because the methods perform very similarly at the beginning, we do not plot the initial part of the curves, that is, we omit points for which the number of function evaluations is less than 100.

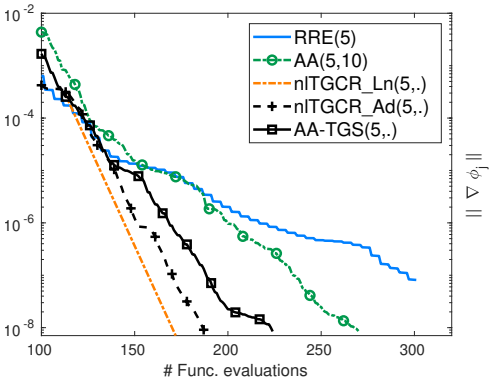
Because `nlTGCR_Ln` is in effect an inexact Newton method, one can expect a superlinear convergence if a proper strategy is adopted when solving the Jacobian systems. These systems are solved inexactly by requiring that we reduce the (linear) residual norm by a certain tolerance τ where τ is adapted. Table 7.1 illustrates the

Table 7.1. The superlinear convergence of nlTGCR_Ln for the Lennard-Jones example.

Its	$\|\nabla E\|_2$	Inner	η
7	1.894e+00	10	4.547e-02
8	6.376e-02	22	1.337e-02
9	3.282e-04	47	1.020e-03
10	7.052e-09	59	2.386e-05



(a)



(b)

Figure 7.1. Comparison of five different methods for the Bratu problem (a) and the Lennard-Jones optimization problem (b).

superlinear convergence of the linear, i.e. inexact, Newton version of nlTGCR, as observed for the Lennard-Jones problem. The algorithm takes 10 outer (Newton) iterations to converge but we only show the last four iterations (as indicated by the column 'Its'). The second column shows the progress of the norm of the gradient, which is nearly quadratic, as can be seen. The third column shows the number of inner steps needed to reduce the residual norm by η at the given Newton step, where η is shown in the fourth column. This tolerance parameter η is determined according to the Eisenstat–Walker update; see Kelley (1995) for details. This update scheme works by trying to produce a quadratically decreasing residual based on gains made in the previous step.

8. Acceleration methods for machine learning

We conclude this article with a few preliminary thoughts on how acceleration methods might be put to work in a world that is increasingly driven by machine learning (ML) and artificial intelligence (AI). Training deep neural networks models is accomplished with one of a number of known iterative procedures. What sets these procedures apart from their counterparts in physical simulations is their reliance on stochastic approaches that exploit large datasets. This presents a completely new landscape for acceleration methods, one for which they were not originally designed. It is too early to definitively say whether or not acceleration methods will be broadly adopted for ML/AI optimization, but it is certainly time to start investigating what modifications to traditional acceleration approaches might be required to deploy them successfully in this context.

Training an AI model is highly demanding, in terms of both memory and computational power. The idea of resorting to acceleration in deep learning is a rather natural one when considering that standard approaches may require tens of thousands of iterations to converge. Anderson acceleration for deep learning tasks was discussed in a number of recent articles; see e.g. Pasini, Yin, Reshniak and Stoyanov (2021, 2022), Sun *et al.* (2021) and Shi *et al.* (2019), among others. Most of these papers advocate some form of regularization to cope with the varying/stochastic nature of the optimization problem.

Before reviewing the challenges posed by stochastic techniques to acceleration methods, we briefly describe the stochastic gradient descent (SGD) algorithm, one of the simplest methods employed to train DNN models. In spite of its simplicity, SGD is a good representative of iterative optimization algorithms in deep learning, because its use is rather widespread and because it shares the same features as those of the more advanced algorithms.

8.1. Stochastic gradient descent

The classical (deterministic) gradient descent (GD) method for minimizing a convex function $\phi(w)$ with respect to w was mentioned in Section 4.2; see equation (4.10). If ϕ is differentiable, the method consists of taking the iterates

$$w_{j+1} = w_j - \eta_j \nabla\phi(w_j), \tag{8.1}$$

where η_j is a scalar termed the *step-size* or *learning rate* in machine learning. The above algorithm is well understood for functions ϕ that are convex. In this situation, the step-size η_j is usually determined by performing a line search, i.e. by selecting the scalar η so as to minimize, or reduce in specific ways, the cost function $\phi(w_j - \eta\nabla\phi(w_j))$ with respect to η. It was Cauchy (1847) who invented the method for solving systems of equations; see also Petrova and Solov'ev (1997) for details on the origin of the method.

In data-related applications, w is a vector of weights needed to optimize a process. This often amounts to finding the best parameters to use, say in a classification

method, so that the value of ϕ at sample points matches some given result across items of a dataset according to a certain measure. Here we simplified notation by writing $\phi(w)$ instead of the more accurate $\phi(x|w)$, which is to be read as 'the value of ϕ for x given the parameter set w'.

In the specific context of deep learning, $\phi(w)$ is often the sum of a large number of other cost functions, that is, we often have

$$\phi(w) = \sum_{i=1}^{N} \phi_i(w) \quad \rightarrow \quad \nabla\phi(w) = \sum_{i=1}^{N} \nabla\phi_i(w). \tag{8.2}$$

The index i here refers to the samples in the training data. In the simplest case where a *mean square error* (MSE) cost function is employed, we would have

$$\phi(w) = \sum_{i=1}^{n} \|\hat{y}_i - \phi_w(x_i)\|_2^2, \tag{8.3}$$

where $\hat{y}_i$ is the target value for item x_i and $y_i = \phi_w(x_i)$ is the result of the model for x_i given the weights represented by w.

Stochastic gradient descent (SGD) methods are designed specifically for the common case where $\phi(w)$ is of the form (8.2). It is usually expensive to compute $\nabla\phi$ but inexpensive to compute a component $\nabla\phi_i(w)$. As a result, the idea is to replace the gradient $\nabla\phi(w)$ in the gradient descent method by $\nabla\phi_{i_j}(w_j)$, where i_j is drawn at random at step j. The result is an iteration of the type

$$w_{j+1} = w_j - \eta_j \nabla\phi_{i_j}(w_j). \tag{8.4}$$

The step-size η_j, called the 'learning rate' in this context, is rarely selected by a line search but it is determined adaptively or set to a constant. Convergence results for SGD have been established in the convex case; see Bubeck (2015), Robbins and Monro (1951), Hardt, Recht and Singer (2016) and Gower *et al.* (2019). The main point, going back to the seminal work by Robbins and Monro (1951), is that when gradients are inaccurately computed, then if they are selected from a process whose noise has a mean of zero then the process will converge in probability to the root.

A straightforward SGD approach that uses a single function ϕ_{i_j} at a time is seldom used in practice because this typically results in a convergence that is too slow. A common middle ground solution between the one-subfunction SGD and the full gradient descent algorithm is to resort to *mini-batching*. In short, the idea consists of replacing the single function ϕ_{i_j} by an average of few such functions, again drawn at random from the full set.

Thus mini-batching begins by partitioning the set $\{1, 2, \ldots, N\}$ into n_B 'mini-batches' $\mathcal{B}_j$, $j = 1, \ldots, n_B$, where

$$\bigcup_{j=1}^{n_B} \mathcal{B}_j = \{1, 2, \ldots, N\}, \quad \text{with } \mathcal{B}_j \bigcap \mathcal{B}_k = \emptyset \quad \text{for } j \neq k.$$

Here, each $\mathcal{B}_j \subseteq \{1, 2, \ldots, n\}$ is a small set of indices. Then, instead of considering a single function ϕ_i, we will consider

$$\phi_{\mathcal{B}_j}(w) \equiv \frac{1}{|\mathcal{B}_j|} \sum_{k \in \mathcal{B}_j} \phi_k(w). \tag{8.5}$$

We will cycle through all mini-batches $\mathcal{B}_j$ of functions, each time performing a group-gradient step of the form

$$w_{j+1} = w_j - \eta \nabla \phi_{\mathcal{B}_j}(w_j), \quad j = 1, 2, \ldots, n_B. \tag{8.6}$$

If each set $\mathcal{B}_j$ is small enough, computing the gradient will be manageable and computationally efficient. One sweep through the whole set of functions as in (8.6) is termed an *epoch*. The number of iterations of SGD and other optimization learning in deep learning is often measured in terms of epochs. It is common to select the partition at random at each new epoch by reshuffling the set $\{1, 2, \ldots, N\}$ and redrawing the batches consecutively from the resulting shuffled set. Models can be expensive to train: the more complex models often require thousands or tens of thousands of epochs to converge.

Mini-batch processing in the random fashion described above is advantageous from a computational point of view since it typically leads to fewer sweeps through each function to achieve convergence. It is also mandatory if we wish to avoid reaching local minima and overfitting. Stochastic approaches of the type just described are at the heart of optimization techniques in deep learning.

8.2. Acceleration methods for deep learning: The challenges

Suppose we want to apply some form of acceleration to the sequence generated by the batched gradient descent iteration (8.6). There is clearly an issue in that the *function changes at each step* by the nature of the stochastic approach. Indeed, by the definition (8.5), it is as if we are trying to find the minimum of a new function at each new step, namely the function (8.5), which depends on the batch $\mathcal{B}_j$. We could use the full gradient, which amounts to using a full batch, i.e. the whole data set, at each step. However, it is often argued that in deep learning an exact minimization of the objective function using the full dataset at once is not only difficult but also counter-productive. Indeed, mini-batching serves other purposes than just better scalability. For example, it helps prevent 'overfitting': using all the data samples at once is similar to interpolating a function in the presence of noise at all the data points. Randomization also helps the process escape from bad local minima.

This brings us to the second problem, namely the lack of convexity of the objective functions invoked in deep learning. This means that all methods that feature a second-order character, such as a quasi-Newton approach, will have both theoretical and practical difficulties. As was seen earlier, Anderson acceleration can be viewed as a secant method similar to a quasi-Newton approach. These methods will potentially utilize many additional vectors but result in little or no acceleration,

if not in a breakdown caused by the non-SPD nature of the Hessian. The lack of convexity and the fact that the problem is heavily over-parametrized means that there are many solutions to which the algorithms can converge. Will acceleration lead to a better solution than that of the baseline algorithm being accelerated? If we consider only the objective function as the sole criterion, one may think that the answer is clear: the lower the better. However, practitioners in this field are more interested in 'generalization' or the property to obtain good classification results on data that is not among the training data set. The problem of generalization has been the object of numerous studies; see e.g. Zhang *et al.* (2021), Li *et al.* (2018), Wu, Zhu and E (2017) and Zhou *et al.* (2020), among many others. Zhang *et al.* (2021) show by means of experiments that looking at DL from the angle of minimizing the loss function fails to explain generalization properties. They show that they can achieve a perfect loss of zero in training models on well-known datasets (MNIST, CIFAR10) that have been modified by randomly changing all labels. In other words, one can obtain parameters whose loss function is minimum but with the worst possible generalization, since the resulting classification would be akin to assigning a random label to each item. A number of other papers (Zhou *et al.* 2020, Neyshabur, Tomioka and Srebro 2015, Li *et al.* 2018, Wu *et al.* 2017) have explored this issue further by attempting to explain generalization with the help of the 'loss landscape', the geometry of the loss function in high-dimensional space. What can be understood from these works is that the problem is far more complex than just minimizing a function. There are many minima, and some are better than others. A local minimum that has a smaller loss function will not necessarily lead to better inference accuracy, and the random character of the learning algorithms plays a central role in achieving a good generalization. This suggests that we should study mechanisms that incorporate or encourage randomness. An illustration will be provided in the next section.

The third challenge is that acceleration methods tend to be memory-intensive, requiring storage of possibly tens of additional vectors to be effective. In deep learning this is not an affordable option. For example, a model such as ChatGPT-3 has 175B parameters, while the more recent Llama 3 involves 450B parameters. This is the main reason why simple methods such as SGD or Adam (Kingma and Ba 2015) are favoured in this context.

8.3. Adapting acceleration methods for ML

Given the discussion of the previous section, one may ask what benefits can be obtained by incorporating second-order information in stochastic methods. Indeed, Bottou, Curtis and Nocedal (2018) discuss the applicability of second-order methods for machine learning and, citing previous work, point out that the convergence rate of a quasi-Newton-type 'stochastic iteration . . . cannot be faster than sublinear'. However, Bottou and Le Cun (2005) state that if the Hessian approximation converges to the exact Hessian at the limit, then the rate of convergence of

SGD is independent of the conditioning of the Hessian. In other words, second-order information makes the method 'better equipped to cope with ill-conditioning than SGD'. Thus careful successive rescaling based on (approximate) second-order derivatives has been successfully exploited to improve convergence of stochastic approaches.

The above discussion may lead the potential researcher to dismiss all acceleration methods for deep learning tasks. There are a few simple variations to acceleration schemes to cope with some of the issues raised in Section 8.2. For example, if our goal is to accelerate the iteration (8.6) with a constant learning rate η, then we could introduce inner iterations to 'iterate within the same batch'. What this means is that we force the acceleration method to act on the same batch for a given number of inner iterations. For example, if we use RRE (see Section 2.6), we could decide to restart every k iterations, each of which is with the same batch $\mathcal{B}_j$. When the next batch is selected, we replace the latest iterate with the result of the accelerated sequence. We found with simple experiments that a method such as RRE will work very poorly without such a scheme.

On the other hand, if we are to embrace a more randomized viewpoint, we could adopt a mixing mechanism whereby the subspaces used in the secant equation evolve across different batches. In other words we no longer force the accelerator to work only on the same batch. Thus the columns of the $\mathcal{X}_j, \mathcal{F}_j$ in (6.5) of Anderson acceleration are now allowed to be associated with different batches. In contrast with RRE, our preliminary experiments show that for AA and AA-TGS this in fact works better than adding an inner loop.

Here is a very simple experiment carried out with the help of the PyTorch library (Paszke *et al.* 2019). We train a small multilayer perceptron (MLP) model (see e.g. Murphy 2022), with two hidden layers to recognize handwritten images from the dataset MNIST (60 000 samples of images of handwritten digits for training, 10 000 samples for testing). We tested four baseline standard methods available in PyTorch: SGD, Adam, RMSprop and AdaGrad. Each of these is then accelerated with RRE (`RRE`), Anderson acceleration (`AAc`) or Anderson-TGS (`TGS`). Here we show the results with SGD only. AA and AA-TGS use a window size of 3 and a restart dimension of 10. Both implement the batch-overlapping subspaces discussed above (no inner loop). In contrast, RRE incorporates an inner loop of five steps (without which it performs rather poorly). The restart dimension for RRE is also set to 5, to match the number of inner steps. We train the model five times using a different randomly drawn subset of 2000 items (out of the 60 000 MNIST training set) each time. With each of the five runs we draw a test sample of 200 (out of the 10 000 MNIST test set) to test the accuracy of the trained model. The accuracy is then averaged across the five runs. For each run the accuracy is measured as just 100 times the ratio of the number of correctly recognized digits over the total in the test set (200). We show the loss function for each of the training algorithms in Figure 8.1(a). Two versions of the baseline algorithm (SGD) are tested, both of which use the same learning rate $\eta = 0.001$. The first, labelled `SGD_bas`, is just the

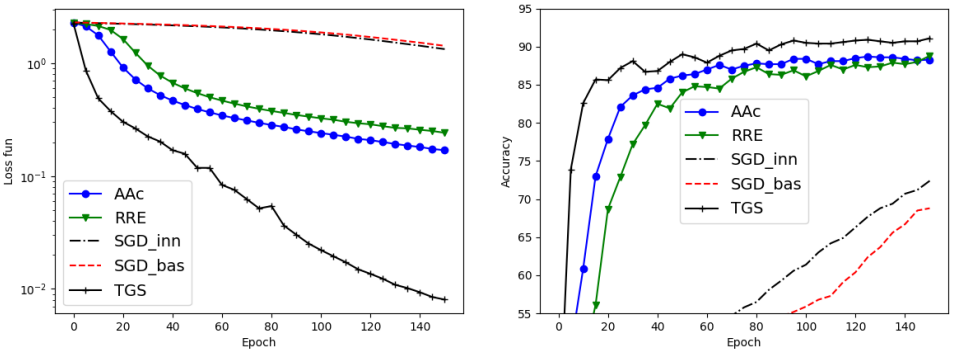


Figure 8.1. Comparison of various techniques for a simple MLP model on the set MNIST.

standard SGD, while the second, `SGD_inn`, incorporates the same inner iteration scheme as RRE (five inner steps). This performs slightly better than the original scheme, sometimes markedly better than with the other optimizers.

One can see a big improvement in the convergence of all the accelerated algorithms. Other tests indicated that adding an inner loop to avoid mixing the subspaces from different batches is detrimental to both AA and AA-TGS, especially when comparing the accuracy. While the improvement in accuracy is significant, we should point out that SGD has not yet fully converged, as we limited the number of epochs to 150. In other tests, e.g. with Adam, we often saw a small improvement in accuracy but not as pronounced. However, in all cases, the accelerated algorithms reach a higher precision much earlier than the baseline methods.

The second remedy addresses the issue of memory. When training AI models, we are limited to a very small number of vectors that we can practically store. So methods such as a restarted RRE, or Anderson, must utilize a very small window size. Note that a window size of m will require a total of $\approx 2m$ vectors for most methods we have seen. This is the reason why a method such as AA-TGS or nlTGCR can be beneficial in deep learning. For both methods, there is a simplification in the linear symmetric case, as was seen earlier. This means that in this special situation a very small window size ($m = 3$ for AA-TGS, $m = 2$ for nlTGCR) will essentially be equivalent to a window size of $m = \infty$, potentially leading to a big advantage. In a general nonlinear optimization situation the Hessian is symmetric, so when the iterates are near the optimum and a nearly linear regime sets in, then the process should benefit from the short-term recurrences of AA-TGS and nlTGCR. In fact this may explain why AA-TGS does so much better at reducing the loss than AA with the same parameters in the previous experiment; see Figure 8.1. From this perspective, any iteration involving a short-term recurrence is worth exploring for machine learning.

What is clear is that acceleration methods of the type discussed in this paper have the potential to emerge as powerful tools for training AI models. While adapting them to the stochastic framework poses challenges, it also offers a promising opportunity for future research.

Acknowledgements

This work would not have been possible without the invaluable collaborations I have had with several colleagues and students, both past and ongoing. I would like to extend special thanks to the following individuals: Claude Brezinski, Michela Redivo-Zaglia, Yuanzhe Xi, Abdelkader Baggag, Ziyuan Tang and Tianshi Xu. I am also grateful for the support provided by the National Science Foundation (grant DMS-2208456).

References

A. Aitken (1926), On Bernoulli's numerical solution of algebraic equations, *Proc. Roy. Soc. Edinburgh* **46**, 289–305.

D. G. Anderson (1965), Iterative procedures for non-linear integral equations, *Assoc. Comput. Mach.* **12**(547), 547–560.

O. Axelsson (1980), Conjugate gradient type-methods for unsymmetric and inconsistent systems of linear equations, *Linear Algebra Appl.* **29**, 1–16.

A. Blair, N. Metropolis, J. von Neumann, A. H. Taub and M. Tsingou (1959), A study of a numerical solution to a two-dimensional hydrodynamical problem, *Math. Comp.* **13**, 145–184.

L. Bottou and Y. Le Cun (2005), On-line learning for very large data sets, *Appl. Stoch. Models Bus. Ind.* **21**, 137–151.

L. Bottou, F. Curtis and J. Nocedal (2018), Optimization methods for large-scale machine learning, *SIAM Rev.* **60**, 223–311.

C. Brezinski (1975), Généralisation de la transformation de Shanks, de la table de Padé et de l'ε-algorithme, *Calcolo* **12**, 317–360.

C. Brezinski (1977), *Accélération de la Convergence en Analyse Numérique*, Vol. 584 of Lecture Notes in Mathematics, Springer.

C. Brezinski (1980), *Padé-Type Approximation and General Orthogonal Polynomials*, Birkhäuser.

C. Brezinski (2000), Convergence acceleration during the 20th century, *J. Comput. Appl. Math.* **122**, 1–21.

C. Brezinski and M. Redivo-Zaglia (1991), *Extrapolation Methods: Theory and Practice*, North-Holland.

C. Brezinski and M. Redivo-Zaglia (2019), The genesis and early developments of Aitken's process, Shanks' transformation, the ϵ-algorithm, and related fixed point methods, *Numer. Algorithms* **80**, 11–133.

C. Brezinski and M. Redivo-Zaglia (2020), *Extrapolation and Rational Approximation: The Works of the Main Contributors*, Springer.

C. Brezinski, S. Cipolla, M. Redivo-Zaglia and Y. Saad (2022), Shanks and Anderson-type acceleration techniques for systems of nonlinear equations, *IMA J. Numer. Anal.* **42**, 3058–3093.

C. Brezinski, M. Redivo-Zaglia and A. Salam (2023), On the kernel of vector ϵ-algorithm and related topics, *Numer. Algorithms* **92**, 207–221.
P. N. Brown and Y. Saad (1990), Hybrid Krylov methods for nonlinear systems of equations, *SIAM J. Sci. Statist. Comput.* **11**, 450–481.
P. N. Brown and Y. Saad (1994), Convergence theory of nonlinear Newton–Krylov algorithms, *SIAM J. Optim.* **4**, 297–330.
S. Bubeck (2015), Convex optimization: Algorithms and complexity, *Found. Trends Mach. Learn.* **8**, 231–357.
S. Cabay and L. W. Jackson (1976), A polynomial extrapolation method for finding limits and antilimits of vector sequences, *SIAM J. Numer. Anal.* **13**, 734–752.
A. L. Cauchy (1847), Methode générale pour la résolution des systèmes d'équations simultanées, *Comp. Rend. Acad. Sci. Paris* **25**, 536–538.
M. Chupin, M.-S. Dupuy, G. Legendre and E. Séré (2021), Convergence analysis of adaptive DIIS algorithms with application to electronic ground state calculations, *ESAIM Math. Model. Numer. Anal.* **55**, 2785–2825.
J. Degroote, K.-J. Bathe and J. Vierendeels (2009), Performance of a new partitioned procedure versus a monolithic procedure in fluid–structure interaction, *Computers & Structures* **87**(11), 793–801.
R. S. Dembo, S. C. Eisenstat and T. Steihaug (1982), Inexact Newton methods, *SIAM J. Numer. Anal.* **18**, 400–408.
P. A. M. Dirac (1929), Quantum mechanics of many-electron systems, *Proc. R. Soc. Lond.* **A123**, 714–733.
R. P. Eddy (1979), Extrapolation to the limit of a vector sequence, in *Information Linkage Between Applied Mathematics and Industry* (P. C. C. Wang, ed.), Academic Press, pp. 387–396.
R. P. Eddy and P. C. C. Wang (1979), Extrapolating to the limit of a vector sequence, in *Information Linkage between Applied Mathematics and Industry*, Academic Press, pp. 387–396.
S. C. Eisenstat and H. F. Walker (1994), Globally convergent inexact Newton methods, *SIAM J. Optim.* **4**, 393–422.
S. C. Eisenstat, H. C. Elman and M. H. Schultz (1983), Variational iterative methods for nonsymmetric systems of linear equations, *SIAM J. Numer. Anal.* **20**, 345–357.
C. Evans, S. Pollock, L. G. Rebholz and M. Xiao (2020), A proof that Anderson acceleration improves the convergence rate in linearly converging fixed-point methods (but not in those converging quadratically), *SIAM J. Numer. Anal.* **58**, 788–810.
V. Eyert (1996), A comparative study on methods for convergence acceleration of iterative vector sequences, *J. Comput. Phys.* **124**, 271–285.
H. Fang and Y. Saad (2009), Two classes of multisecant methods for nonlinear acceleration, *Numer. Linear Algebra Appl.* **16**, 197–221.
G. E. Forsythe (1951), Gauss to Gerling on relaxation, *Mathematical Tables and Other Aids to Computation* **5**, 255–258.
G. E. Forsythe (1953), Solving linear algebraic equations can be interesting, *Bull. Amer. Math. Soc.* **59**, 299–329.
G. Gamov (1966), *Thirty Years That Shook Physics: The Story of Quantum Theory*, Dover.
B. Germain-Bonne (1978), Estimation de la limite de suites et formalisation de procédés d'accélération de convergence. PhD thesis, Université des Sciences et Techniques de Lille.

G. H. Golub and C. F. Van Loan (2013), *Matrix Computations*, fourth edition, Johns Hopkins University Press.
G. H. Golub and R. S. Varga (1961), Chebyshev semi-iterative methods, successive over-relaxation iterative methods, and second order Richardson iterative methods, *Numer. Math.* **3**, 157–168.
R. M. Gower, N. Loizou, X. Qian, A. Sailanbayev, E. Shulgin and P. Richtárik (2019), SGD: General analysis and improved rates, in *International Conference on Machine Learning*, PMLR, pp. 5200–5209.
R. Haelterman, J. Degroote, D. V. Heule and J. Vierendeels (2010), On the similarities between the quasi-Newton inverse least squares method and GMRES, *SIAM J. Numer. Anal.* **47**, 4660–4679.
M. Hardt, B. Recht and Y. Singer (2016), Train faster, generalize better: Stability of stochastic gradient descent, in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds), Vol. 48 of Proceedings of Machine Learning Research, PMLR, pp. 1225–1234.
H. He, Z. Tang, S. Zhao, Y. Saad and Y. Xi (2024), nlTGCR: A class of nonlinear acceleration procedures based on conjugate residuals, *SIAM J. Matrix Anal. Appl.* **45**, 712–743.
M. R. Hestenes and E. L. Stiefel (1952), Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Standards* **49**, 409–436.
M. R. Hestenes and J. Todd (1991), Mathematicians learning to use computers. NIST Special Publication 730, NBS-INA – The Institute for Numerical Analysis – UCLA 1947–1954.
N. J. Higham and N. Strabić (2016), Anderson acceleration of the alternating projections method for computing the nearest correlation matrix, *Numer. Algorithms* **72**, 1021–1042.
J. Jacobsen and K. Schmitt (2002), The Liouville–Bratu–Gelfand problem for radial operators, *J. Differential Equations* **184**, 283–298.
K. Jbilou (1988), Méthodes d'extrapolation et de projection: Applications aux suites de vecteurs. PhD thesis, Université des Sciences et Techniques de Lille.
K. Jbilou and H. Sadok (1991), Some results about vector extrapolation methods and related fixed point iteration, *J. Comput. Appl. Math.* **36**, 385–398.
K. Jbilou and H. Sadok (2000), Vector extrapolation methods: Application and numerical comparison, *J. Comput. Appl. Math* **122**, 149–165.
K. C. Jea and D. M. Young (1980), Generalized conjugate gradient acceleration of non-symmetrizable iterative methods, *Linear Algebra Appl.* **34**, 159–194.
S. Kaniel and J. Stein (1974), Least-square acceleration of iterative methods for linear equations, *J. Optim. Theory Appl.* **14**, 431–437.
C. T. Kelley (1995), *Iterative Methods for Linear and Nonlinear Equations*, Vol. 16 of Frontiers and Applied Mathematics, SIAM.
T. Kerkhoven and Y. Saad (1992), Acceleration techniques for decoupling algorithms in semiconductor simulation, *Numer. Math.* **60**, 525–548.
D. P. Kingma and J. Ba (2015), Adam: A method for stochastic optimization, in *3rd International Conference on Learning Representations (ICLR 2015)*. Conference Track Proceedings.
C. Kittel (1986), *Introduction to Solid State Physics*, Wiley.
W. Kohn and L. J. Sham (1965), Self-consistent equations including exchange and correlation effects, *Phys. Rev.* **140**, A1133–A1138.

C. Lanczos (1950), An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, *J. Res. Nat. Bur. Standards* **45**, 255–282.
C. Lanczos (1952), Solution of systems of linear equations by minimized iterations, *J. Res. Nat. Bur. Standards* **49**, 33–53.
H. Li, Z. Xu, G. Taylor, C. Studer and T. Goldstein (2018), Visualizing the loss landscape of neural nets, in *Advances in Neural Information Processing Systems 31* (S. Bengio *et al.*, eds), Curran Associates, pp. 6391–6401.
M. Mešina (1977), Convergence acceleration for the iterative solution of the equations $X = AX + f$, *Comput. Methods Appl. Mech. Engrg* **10**, 165–173.
G. Meurant and J. D. Tebbens (2020), *Krylov Methods for Nonsymmetric Linear Systems: From Theory to Computations*, Vol. 57 of Springer Series in Computational Mathematics, Springer.
L. Meyer, C. Barrett and P. Haasen (1964), New crystalline phase in solid argon and its solid solutions, *J. Chem. Phys.* **40**, 2744–2745.
A. Mohsen (2014), A simple solution of the Bratu problem, *Comput. Math. Appl.* **67**, 26–33.
K. P. Murphy (2022), *Probabilistic Machine Learning: An Introduction*, MIT Press.
Y. Nesterov (2014), *Introductory Lectures on Convex Optimization: A Basic Course*, first edition, Springer.
B. Neyshabur, R. Tomioka and N. Srebro (2015), In search of the real inductive bias: On the role of implicit regularization in deep learning, in *3rd International Conference on Learning Representations (ICLR), Workshop Track Proceedings* (Y. Bengio and Y. LeCun, eds).
J. M. Ortega and W. C. Rheinbolt (1970), *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press.
C. C. Paige (1971), The computation of eigenvalues and eigenvectors of very large sparse matrices. PhD thesis, Institute of Computer Science, University of London.
C. C. Paige (1980), Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem, *Linear Algebra Appl.* **34**, 235–258.
M. L. Pasini, J. Yin, V. Reshniak and M. Stoyanov (2021), Stable Anderson acceleration for deep learning. Available at https://dblp.org/rec/journals/corr/abs-2110-14813.bib.
M. L. Pasini, J. Yin, V. Reshniak and M. K. Stoyanov (2022), Anderson acceleration for distributed training of deep learning models, in *SoutheastCon 2022*, IEEE, pp. 289–295.
A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.* (2019), PyTorch: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems 32* (H. Wallach *et al.*, eds), Curran Associates.
S. S. Petrova and A. D. Solov'ev (1997), The origin of the method of steepest descent, *Hist. Math.* **24**, 361–375.
S. Pollock and L. G. Rebholz (2021), Anderson acceleration for contractive and noncontractive operators, *IMA J. Numer. Anal.* **41**, 2841–2872.
B. Pugachëv (1977), Acceleration of the convergence of iterative processes and a method of solving systems of non-linear equations, *USSR Comput. Math. Math. Phys.* **17**, 199–207.
P. Pulay (1980), Convergence acceleration of iterative sequences. the case of SCF iteration, *Chem. Phys. Lett.* **73**, 393–398.
P. Pulay (1982), Improved SCF convergence acceleration, *J. Comput. Chem.* **3**, 556–560.

I. Ramière and T. Helfer (2015), Iterative residual-based vector methods to accelerate fixed point iterations, *Comput. Math. Appl.* **70**, 2210–2226.
J. K. Reid (1971), On the method of conjugate gradients for the solution of large sparse systems of linear equations, in *Large Sparse Sets of Linear Equations* (J. K. Reid, ed.), Academic Press, pp. 231–254.
L. F. Richardson (1910), The approximate arithmetical solution by finite differences of physical problems involving differential equations with an application to the stresses to a masonry dam, *Philos. Trans. Roy. Soc. A* **210**, 307–357.
H. Robbins and S. Monro (1951), A stochastic approximation method, *Ann. Math. Statist.* pp. 400–407.
T. Rohwedder (2010), An analysis for some methods and algorithms of quantum chemistry. PhD thesis, Technische Universität Berlin, Fakultät II - Mathematik und Naturwissenschaften.
W. Romberg (1955), Vereinfachte numerische Integration, *Norske Vid. Selsk. Forh.* **28**, 30–36.
Y. Saad (2003), *Iterative Methods for Sparse Linear Systems*, second edition, SIAM.
Y. Saad (2011), *Numerical Methods for Large Eigenvalue Problems*, Vol. 66 of Classics in Applied Mathematics, SIAM.
Y. Saad (2022), The origin and development of Krylov subspace methods, *Comput. Sci. Engrg* **24**, 28–39.
Y. Saad and M. H. Schultz (1986), GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* **7**, 856–869.
Y. Saad, J. Chelikowsky and S. Shontz (2009), Numerical methods for electronic structure calculations of materials, *SIAM Rev.* **52**, 3–54.
D. Shanks (1955), Non-linear transformations of divergent and slowly convergent sequences, *J. Math. Phys.* **34**, 1–42.
J. W. Sheldon (1955), On the numerical solution of elliptic difference equations, *Mathematical Tables and Other Aids to Computation* **9**, 101–112.
W. Shi, S. Song, H. Wu, Y.-C. Hsu, C. Wu and G. Huang (2019), Regularized Anderson acceleration for off-policy deep reinforcement learning, in *Advances in Neural Information Processing Systems 32* (H. Wallach *et al.*, eds), Curran Associates.
G. H. Shortley (1953), Use of Tschebyscheff-polynomial operators in the solution of boundary value problems, *J. Appl. Phys.* **24**, 392–396.
A. Sidi (2012), Review of two vector extrapolation methods of polynomial type with applications to large-scale problems, *J. Comput. Sci.* **3**, 92–101.
A. Sidi, W. F. Ford and D. A. Smith (1986), Acceleration of convergence of vector sequences, *SIAM J. Numer. Anal.* **23**, 178–196.
D. A. Smith, W. F. Ford and A. Sidi (1987), Extrapolation methods for vector sequences, *SIAM Rev.* **29**, 199–233.
K. Sun, Y. Wang, Y. Liu, B. Pan, S. Jui, B. Jiang, L. Kong *et al.* (2021), Damped Anderson mixing for deep reinforcement learning: Acceleration, convergence, and stabilization, in *Advances in Neural Information Processing Systems 34* (M. Ranzato *et al.*, eds), Curran Associates, pp. 3732–3743.
Z. Tang, T. Xu, H. He, Y. Saad and Y. Xi (2024), Anderson acceleration with truncated Gram–Schmidt, *SIAM J. Matrix Anal. Appl.* **45**, 1850–1872.
A. Toth and C. T. Kelley (2015), Convergence analysis for Anderson acceleration, *SIAM J. Numer. Anal.* **53**, 805–819.

D. Vanderbilt and S. G. Louie (1984), Total energies of diamond (111) surface reconstructions by a linear combination of atomic orbitals method, *Phys. Rev. B* **30**, 6118–6130.

P. K. W. Vinsome (1976), ORTHOMIN: An iterative method for solving sparse sets of simultaneous linear equations, in *Proceedings of the Fourth Symposium on Reservoir Simulation*, Society of Petroleum Engineers of AIME, pp. 149–159.

H. F. Walker and P. Ni (2011), Anderson acceleration for fixed-point iterations, *SIAM J. Numer. Anal.* **49**, 1715–1735.

L. Wu, Z. Zhu and W. E (2017), Towards understanding generalization of deep learning: Perspective of loss landscapes. Available at https://dblp.org/rec/journals/corr/WuZE17.bib.

P. Wynn (1956), On a device for computing the $e_m(s_n)$ transformation, *Mathematical Tables and Other Aids to Computation* **10**, 91–96.

P. Wynn (1962), Acceleration techniques for iterated vector and matrix problems, *Math. Comp.* **16**, 301–322.

D. Young (1954), On Richardson's method for solving linear systems with positive definite matrices, *J. Math. Phys.* **32**, 243–255.

C. Zhang, S. Bengio, M. Hardt, B. Recht and O. Vinyals (2021), Understanding deep learning (still) requires rethinking generalization, *Commun. Assoc. Comput. Mach.* **64**, 107–115.

J. Zhang, B. O'Donoghue and S. Boyd (2020), Globally convergent type-I Anderson acceleration for nonsmooth fixed-point iterations, *SIAM J. Optim.* **30**, 3170–3197.

P. Zhou, J. Feng, C. Ma, C. Xiong, S. C. H. Hoi and W. E (2020), Towards theoretically understanding why SGD generalizes better than Adam in deep learning, in *Advances in Neural Information Processing Systems 33* (H. Larochelle *et al.*, eds), Curran Associates, pp. 21285–21296.