SIAM J. MATRIX ANAL. APPL.
Vol. 45, No. 1, pp. 712–743

nlTGCR: A CLASS OF NONLINEAR ACCELERATION PROCEDURES BASED ON CONJUGATE RESIDUALS*

HUAN HE†, ZIYUAN TANG‡, SHIFAN ZHAO§, YOUSEF SAAD‡, AND YUANZHE XI§

Abstract. This paper develops a new class of nonlinear acceleration algorithms based on extending conjugate residual-type procedures from linear to nonlinear equations. The main algorithm has strong similarities with Anderson acceleration as well as with inexact Newton methods—depending on which variant is implemented. We prove theoretically and verify experimentally, on a variety of problems from simulation experiments to deep learning applications, that our method is a powerful accelerated iterative algorithm. The code is available at https://github.com/Data-driven-numerical-methods/Nonlinear-Truncated-Conjugate-Residual.

Key words. nonlinear acceleration, generalized conjugate residual, truncated GCR, Anderson acceleration, Newton's method, deep learning

MSC codes. 65F10, 68W25, 65F08, 90C53

DOI. 10.1137/23M1576360

1. Introduction.

There has been a surge of interest in recent years in numerical algorithms whose goal is to accelerate iterative schemes for solving the following problem:

$$\text{Find } x \in \mathbb{R}^n \text{ such that } f(x) = 0, \tag{1.1}$$

where f is a continuously differentiable mapping from $\mathbb{R}^n$ to $\mathbb{R}^n$. This problem can itself originate from unconstrained optimization where we need to minimize a scalar function,

$$\min_x \phi(x), \tag{1.2}$$

in which $\phi : \mathbb{R}^n \to \mathbb{R}$. In this situation, we will be interested in a local minimum which can be found as a zero of the system of equations $f(x) = 0$ where $f(x) = \nabla\phi(x)$.

The problem (1.1) can be formulated as a *fixed point* problem, where one seeks to find the fixed point of a mapping g from $\mathbb{R}^n$ to itself:

$$\text{Find } x \in \mathbb{R}^n \text{ such that } x = g(x). \tag{1.3}$$

This can be achieved by setting, for example, $g(x) = x + \beta f(x)$ for some nonzero β. Given a mapping g, the related *fixed point iteration,* i.e., the sequence generated by

$$x_{j+1} = g(x_j), \tag{1.4}$$

*Received by the editors May 31, 2023; accepted for publication (in revised form) by U. M. Yang November 15, 2023; published electronically February 29, 2024.
https://doi.org/10.1137/23M1576360
Funding: The work of the second and fourth authors was supported by National Science Foundation award DMS-2208456. The work of the third and fifth authors was supported by National Science Foundation award DMS-2208412.
†Department of Computer Science, Emory University, Atlanta, GA 30322 USA (hehuannb@gmail.com).
‡Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455 USA (tang0389@umn.edu, saad@umn.edu).
§Department of Mathematics, Emory University, Atlanta, GA 30322 USA (shifan.zhao@emory.edu, yxi26@emory.edu).

may converge to the fixed point of (1.3) and when $g(x) = x + \beta f(x)$ then this limit is clearly a solution to the problem (1.1). However, fixed point iterations often converge slowly, or may even diverge. As a result *acceleration methods* are often invoked to improve their convergence or to establish it.

A number of such acceleration methods have been proposed in the past. It is important to clarify the terminology and discuss the distinction between *acceleration methods,* which aim at accelerating the convergence of a fixed point sequence of the form (1.4), and *solvers,* which aim at finding solutions to (1.1). Among acceleration techniques are "extrapolation-type" algorithms such as the reduced-rank extrapolation (RRE) [69], the minimal-polynomial extrapolation [15], the modified minimal-polynomial extrapolation [38], and the vector ϵ-algorithms [10]. These typically produce a new sequence from the original one by combining them without invoking the mapping g in the process. Another class of methods produce a new sequence by utilizing both the iterates and the mapping g. Among these, Anderson acceleration (AA) [3] has received enormous attention in recent years due to its success in solving a wide range of problems [67, 30, 68, 19, 71, 56, 36, 33, 77, 37]. AA can be seen as an inexpensive alternative to second order methods such as quasi-Newton type techniques. It is often used quite successfully without global convergence strategies such as line search or trust-region techniques. These advantages made the method popular in applications ranging from quantum physics, where they were first developed, to machine learning. We refer readers to [11] for a survey of acceleration methods.

As was stated above, it is clear that one can invoke one of these accelerators for solving (1.1) by applying it to the fixed point sequence associated with $g(x) = x + \beta f(x)$. AA and its sibling Pulay mixing [59, 60] were devised precisely in this way. Thus, *an accelerator of this type can be viewed as a solver,* in the same way that a linear accelerator (e.g., conjugate gradient) combined with some basic iteration, such as Richardson, can be viewed as a "solver." This class of techniques does not include the extrapolation-type methods discussed above because *they require the computation of $g(x)$ for an arbitrary x.* In extrapolation methods we have a sequence of vectors, but we do not have access to the function g for evaluating $g(x)$.

We can also ask the question of whether or not a given solver can be viewed as an accelerator. If the solver only requires evaluating $f(x)$ for an arbitrary x, then clearly we can apply it to find the root of the equation $f(x) \equiv x - g(x) = 0$, which requires the computation $g(x)$ given x. The related iterative process can be viewed as an acceleration technique for the fixed point mapping $g(x)$. Thus, our definition of an accelerator is broad and it encompasses any method that aims to speed up a fixed point iteration by requiring only function evaluations at each step.

AA is a good representative of this class of methods. There are three issues with AA, and similar accelerators, which this article aims to address. The first is that for optimization problems, AA does not seem to be amenable to exploiting the symmetry of the Jacobian or Hessian. If we had to solve a linear system with AA, the sequence resulting from the algorithm cannot be written in the form of a short-term recurrence, as is the case with conjugate gradient or conjugate residual algorithms, for example. For nonlinear optimization problems where the Hessian is symmetric, this indicates that AA does not take advantage of symmetry and as such it may become expensive in terms of memory and computational cost. This is especially true in a nonconvex stochastic setting, where a large number of iterates are often needed. This can be an acute problem, particularly in machine learning, where we often encounter practical situations in which the number of parameters is quite large, making it impractical to use a large number of vectors in AA. Although recent works in the literature made efforts to improve the convergence speed of AA [76, 77, 50, 67, 81, 34], they did

not attempt to reduce memory cost. One of the goals of this paper is to propose an acceleration technique that exploits symmetry or near symmetry to reduce the computational cost.

A second problem with AA, which is perhaps a result of its simplicity, is that while it shares some similarities with quasi-Newton methods, it does not exploit standard methods that are common in second order methods to improve *global convergence* characteristics. The inclusion of line search or trust-region methods is necessary if one wishes to solve realistic problems. A secondary goal of this paper is to introduce an AA-like method that implements global convergence strategies borrowed from inexact Newton and quasi-Newton methods.

Finally, one of the issues with AA, and other acceleration methods, is that it uses a crude linear model. Specifically, acceleration methods typically rely on two sets of consecutive vector differences, namely the differences $\Delta f_i \equiv f(x_{i+1}) - f(x_i)$ and the associated $\Delta x_i \equiv x_{i+1} - x_i$. The issue is that their approximate solutions are developed from the relation $\Delta f_i \approx J(x_i)\Delta x_i$, where $J(x_i)$ is the Jacobian at x_i. This linear approximation is likely to be rather inaccurate, especially when the iterate is far from its limit. It is important to develop techniques that will avoid relying on such rough approximations.

Nonlinear acceleration methods of the type discussed in this paper appeared first in the physics literature where they were needed to accelerate very complex and computationally intensive processes, such as the self-consistent field (SCF) iteration. The best-known of these methods was discovered by Anderson [3] in 1965. In the early 1980s, Pulay proposed a similar scheme, which he called direct inversion on the iterative subspace (DIIS) [59, 60]. Both methods were designed specifically for SCF iterations and it turns out that, although formulated differently, AA and DIIS are essentially equivalent; see, e.g., [18, 27, 46]. For this reason, the method is often referred to as "Anderson–Pulay mixing," where mixing in this context refers to the process by which a new fixed point iterate is *mixed* with previous ones to accelerate the process. AA was rediscovered again in a different form in a 2000 paper by Oosterlee and Washio [55], who applied their technique to accelerate nonlinear multigrid iterations.

The link between nonlinear acceleration methods such as AA and secant-type methods was first unraveled by Eyert [26] when he compared AA with a multisecant method proposed by Vanderbilt and Louie more than a decade earlier [73]. The article [61] explored this idea further and expanded on it by proposing two classes of multisecant methods. Thereafter, AA started to be studied and utilized by researchers outside the field of physics; see, e.g., [20, 75, 72, 49, 6, 57, 25], among many others.

The primary contribution of the present paper is to take another look at this class of methods and develop a technique that is derived by a careful extension of a linear iterative method to nonlinear systems. The paper is motivated primarily by a desire to overcome the three weaknesses of AA mentioned earlier and accelerate the stochastic optimization algorithms used in deep learning applications.

2. Background: Inexact Newton, quasi-Newton, and Anderson acceleration. The goal of this section is to clarify key features of the method proposed in this paper in order to establish links with known methods. Many of the approaches developed for solving (1.1) are rooted in Newton's method, which exploits the local linear model:

$$f(x + \Delta x) \approx f(x) + J(x)\Delta x, \tag{2.1}$$

where $J(x)$ is the Jacobian matrix at x.

Notation. We will often refer to an evolving set of columns where the most recent vectors from a sequence are retained. In this situation, we found it convenient to use the following convention. For a given $m \geq 0$ we set

$$j_m = \max\{0, j-m+1\}, \quad m_j = \min\{m, j+1\} \equiv j - j_m + 1. \tag{2.2}$$

2.1. Inexact Newton methods. Newton's method determines $\Delta x_j = x_{j+1} - x_j$ at step j, to make the right-hand side on (2.1) equal to zero when $x = x_j$, which is achieved by solving the Newton linear system $J(x_j)\delta + f(x_j) = 0$. Inexact Newton methods, e.g., [41, 21, 12] among many references, compute a sequence of iterates in which the above Newton systems are solved approximately, often by an iterative method. Given an initial guess x_0, the iteration proceeds as follows:

$$\text{Solve} \qquad J(x_j)\delta_j \approx -f(x_j), \tag{2.3}$$

$$\text{Set} \qquad x_{j+1} = x_j + \delta_j. \tag{2.4}$$

Note that 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 to be $r_j \equiv -f(x_j)$.

The technique for solving the local system (2.3) is not specified. Suppose that we invoke a Krylov subspace method for solving (2.3). If we set $J \equiv J(x_j)$, then the method will usually generate an approximate solution that can be written in the form

$$\delta_j = V_j y_j, \tag{2.5}$$

where V_j is an orthonormal basis of the Krylov subspace

$$K_j = \text{Span}\{r_j, Jr_j, \ldots, J^{m-1}r_j\}. \tag{2.6}$$

The vector y_j represents the expression of the solution in the basis V_j. For example, if GMRES or, equivalently, generalized conjugate residual (GCR) [22] is used, then y_j becomes $y_j = (JV_j)^\dagger(-f(x_j))$, where $\dagger$ denotes the pseudoinverse. In essence the inverse Jacobian is approximated by the rank m matrix:

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

In inexact Newton methods the approximations just defined are valid only for the jth step, i.e., once the solution is updated, they are discarded and the process will essentially compute a new Krylov subspace and related approximate Jacobian at point x_{j+1}. This "lack of memory" can be an impediment to performance. In contrast, quasi-Newton methods will compute approximate Jacobians or their inverses by a process that is continuously being updated, using the most recent iterate for this update.

2.2. Quasi-Newton methods. 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 (2.1) to be satisfied by the updated J_{j+1} which is built at step j. This means that the following *secant condition* is imposed:

$$J_{j+1}\Delta x_j = \Delta f_j, \tag{2.7}$$

where $\Delta f_j := f(x_{j+1}) - f(x_j)$. The following *no-change condition* is also imposed:

$$J_{j+1}q = J_j q \quad \forall q \quad \text{such that} \quad q^T \Delta x_j = 0. \tag{2.8}$$

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 (2.7) and (2.8) and it can be obtained by the update formula:

$$J_{j+1} = J_j + (\Delta f_j - J_j \Delta x_j) \frac{\Delta x_j^T}{\Delta x_j^T \Delta x_j}. \tag{2.9}$$

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 (2.7) becomes

$$G_{j+1} \Delta f_j = \Delta x_j. \tag{2.10}$$

By minimizing $E(G_{j+1}) = \|G_{j+1} - G_j\|_F^2$ with respect to G_{j+1} subject to (2.10), one finds this update formula for the inverse Jacobian,

$$G_{j+1} = G_j + (\Delta x_j - G_j \Delta f_j) \frac{\Delta f_j^T}{\Delta f_j^T \Delta f_j}, \tag{2.11}$$

which is also the only update satisfying both the secant condition (2.10) and the no-change condition for the inverse Jacobian:

$$(G_{j+1} - G_j) q = 0 \quad \forall\ q \perp \Delta f_j. \tag{2.12}$$

We will revisit secant-type methods again when we discuss AA in the next section. 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.

2.3. General nonlinear acceleration and Anderson's method. Acceleration methods, such as AA, take a different viewpoint altogether. Their goal is to accelerate a given fixed point iteration of the form (1.4). Thus, AA starts with an initial x_0 and sets $x_1 = g(x_0) = x_0 + \beta f_0$, where $\beta > 0$ is a parameter. At step $j > m$ we define $X_j = [x_{j-m}, \ldots, x_{j-1}]$ and $F_j = [f_{j-m}, \ldots, f_{j-1}]$ along with the differences:

$$\mathcal{X}_j = [\Delta x_{j-m} \ \cdots \ \Delta x_{j-1}] \in \mathbb{R}^{n \times m}, \qquad \mathcal{F}_j = [\Delta f_{j-m} \ \cdots \ \Delta f_{j-1}] \in \mathbb{R}^{n \times m}. \tag{2.13}$$

We then define the next AA iterate as follows:

$$x_{j+1} = x_j + \beta f_j - (\mathcal{X}_j + \beta \mathcal{F}_j)\ \theta^{(j)} \quad \text{where} \tag{2.14}$$

$$\theta^{(j)} = \operatorname{argmin}_{\theta \in \mathbb{R}^m} \|f_j - \mathcal{F}_j \theta\|_2. \tag{2.15}$$

Note that x_{j+1} can be expressed with the help of intermediate vectors:

$$\bar{x}_j = x_j - \mathcal{X}_j\ \theta^{(j)}, \quad \bar{f}_j = f_j - \mathcal{F}_j\ \theta^{(j)}, \quad x_{j+1} = \bar{x}_j + \beta \bar{f}_j. \tag{2.16}$$

There is an underlying quasi-Newton second order method aspect to the procedure. In Broyden-type methods, Newton's iteration $x_{j+1} = x_j - J(x_j)^{-1} f_j$ is replaced with $x_{j+1} = x_j - G_j f_j$, where G_j approximates the inverse of the Jacobian $J(x_j)$ at x_j by the update formula $G_{j+1} = G_j + (\Delta x_j - G_j \Delta f_j) v_j^T$ in which v_j is defined in different ways; see [61] for details. AA belongs to the class of *multisecant methods.* Indeed, the approximation (2.14) can be written as

$$x_{j+1} = x_j - [-\beta I + (\mathcal{X}_j + \beta \mathcal{F}_j)(\mathcal{F}_j^T \mathcal{F}_j)^{-1} \mathcal{F}_j^T] f_j \equiv x_j - G_j f_j. \tag{2.17}$$

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

$$G_j = G_{j-m} + (\mathcal{X}_j - G_{j-m}\mathcal{F}_j)(\mathcal{F}_j^T \mathcal{F}_j)^{-1}\mathcal{F}_j^T. \tag{2.18}$$

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,[1]

$$G_j \mathcal{F}_j = \mathcal{X}_j. \tag{2.19}$$

This link between AA and Broyden multisecant type updates was first unraveled by Eyert [26] and expanded upon in [61]. 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 1, update is applied at each step.

2.4. The issue of symmetry. Consider again the specific case where the nonlinear function $f(x)$ is the gradient of some scalar function $\phi(x)$ to be minimized, i.e., $f(x) = \nabla\phi(x)$. In this situation the Jacobian of f becomes $\nabla^2\phi$ the Hessian of ϕ, and therefore it is symmetric. Approximate Jacobians that are implicitly or explicitly extracted in the algorithm will be symmetric or nearly symmetric. Therefore this raises the possibility of developing accelerators that take advantage of symmetry or near-symmetry. One way to achieve this is to extend *linear solvers* that take advantage of symmetry to the nonlinear context. This was one of the primary initial motivations for this work.

We saw earlier that AA is a multisecant version of a Broyden type II method where the approximate inverse Jacobian is updated by formula (2.18). An obvious observation here is that the symmetry of the Jacobian cannot be exploited in any way in this formula. This has been considered in the literature (very) recently; see, for example, [7, 66, 8]. In a 1983 report, Schnabel [65] showed that the matrix G_j obtained by a multisecant method that satisfies the secant condition (2.19) is symmetric iff the matrix $\mathcal{X}_j^T \mathcal{F}_j$ is symmetric. It is possible to explicitly force symmetry by employing generalizations of the symmetric versions of Broyden-type methods. Thus, the authors of [7, 8] recently developed a multisecant version of the Powell symmetric Broyden update due to Powell [58] while the article [66] proposed a symmetric multisecant method based on the popular Broyden–Fletcher–Goldfarb–Shanno (BFGS) approach as well as the Davidon–Fletcher–Powell update. However, there are a number of issues with the symmetric versions of multisecant updates, some of which are discussed in [66].

3. Nonlinear truncated generalized conjugate residual (nlTGCR) algorithm. For understanding the conjugate residual–based methods in the nonlinear case, it is important to first provide some background for linear systems. A large class of Krylov subspace methods for solving nonsymmetric linear systems have been developed in the past four decades. The reader is referred to the recent volume by Meurant and Tebbens [52], which contains a rather exhaustive and detailed coverage of these methods. The main aim of the techniques proposed in this article is to adapt the residual-minimizing subclass of Krylov methods for linear systems to the nonlinear case. The guiding principle in this adaption is that we would like it to also approximately minimize the nonlinear residuals. This is in contrast with inexact Newton methods, where the goal is to roughly solve the linear systems that arise in Newton's method as a way to provide a good search direction.

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

Algorithm 3.1 GCR.

1: **Input**: Matrix A, RHS b, initial x_0.
2: Set $p_0 := r_0 \equiv b - Ax_0$.
3: **for** $j = 0, 1, 2, \ldots,$ until convergence **do**
4: $\quad \alpha_j := \langle r_j, Ap_j\rangle / \langle Ap_j, Ap_j\rangle$
5: $\quad x_{j+1} := x_j + \alpha_j p_j$
6: $\quad r_{j+1} := r_j - \alpha_j Ap_j$
7: $\quad p_{j+1} := r_{j+1} - \sum_{i=0}^{j} \beta_{ij} p_i$ where $\beta_{ij} := \langle Ar_{j+1}, Ap_i\rangle / \langle Ap_i, Ap_i\rangle$
8: **end for**

3.1. The linear case: Generalized conjugate residual algorithm. We first consider solving a linear system of the form

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

A number of iterative methods developed in the 1980s aimed at minimizing the norm of the residual $r = b - Ax$ of a new iterate that lies in a Krylov subspace; see [52] for a detailed account. Among these, we focus on the GCR algorithm [22] for solving (3.1), which is sketched in Algorithm 3.1.

The main point of the algorithm is to build a sequence of search directions p_i, $i = 0 : j$ at step j so that the vectors Ap_i are orthogonal. This is done in line 7. With this we know that the iterate as defined by lines 4–5 is optimal in the sense that it yields the smallest residual norm among solution vectors selected from the Krylov subspace $x_0 + \text{Span}\{p_0, \ldots, p_k\}$—see [63, pp. 195–196]. The GCR algorithm is mathematically equivalent[2] to GMRES [64] and to some of the forms of other methods developed earlier, e.g., ORTHOMIN [74], ORTHODIR [39], and Axelsson's CGLS method [5].

Next, we will discuss a truncated version of GCR in which the Ap_i's are orthogonal only to the m previous ones instead of all of them. This algorithm was first introduced by Vinsome as early as 1976 and was named ORTHOMIN [74]. We will just refer to it as the truncated version of GCR or TGCR(m). In a practical implementation, we need to keep a set of m vectors at step j for the p_i's and another set for the vectors $v_i = Ap_i$. In addition, we replace the classical Gram–Schmidt of line 7 of Algorithm 3.1 by the modified form of Gram–Schmidt: the vector Ar_{j+1} initially set to a vector v which is orthogonalized against the successive Ap_i's. Thus, line 7 of Algorithm 3.1 becomes

7a. $\quad p := r_{j+1}$; $v := Ap$; and $j_m := \max(0, j - m + 1)$
7b. $\quad$ **for** $i = j_m : j$
7c. $\quad\quad \beta_{ij} := \langle v, Ap_i\rangle$
7d. $\quad\quad p := p - \beta_{ij} p_i$; $v := v - \beta_{ij} v_i$;
7e. $\quad$ **end for**
7f. $\quad p_{j+1} := p/\|v\|$; $v_{j+1} := v/\|v\|$;

We refer to the algorithm obtained from Algorithm 3.1 where line 7 is replaced by lines 7a–7f above as the truncated GCR (TGCR(m)) algorithm. This algorithm is

[2]Here equivalent is meant in the sense that if exact arithmetic is used and if the compared algorithms both succeed in producing the jth iterate from the same initial x_0, then the two iterates are equal.

a slight variant of the original ORTHOMIN introduced in [74] and analyzed in [22]. It produces a system of vectors $V_{j+1} = [v_{j_m}, v_{j_m+1}, \ldots, v_j, v_{j+1}]$ that is orthonormal. When $j \geq m$, V_{j+1} consists of a "window" of $m+1$ vectors. TGCR(m) approximately minimizes the quadratic form $\phi_q(x) \equiv \frac{1}{2}\|b - Ax\|_2^2$ in a certain Krylov subspace. With $m = \infty$ we obtain the nonrestarted GCR method—which is equivalent to the nonrestarted GMRES [64].

A few properties of the vectors generated in TGCR(m) have been analyzed in [22, Thm. 4.1] which also discussed the convergence of the algorithm when A is positive definite, i.e., when $A+A^T$ is symmetric positive definite (SPD). When A is symmetric a number of simplifications take place in Algorithm 3.1. In this situation, all the β_{ij}'s except β_{jj} vanish. The resulting simplified algorithm yields the standard conjugate residual algorithm which dates back to Stiefel [70]; see [64, section 6.8] for details. See Appendix A for a unified presentation of a number of theoretical results of GCR.

3.2. The nonlinear extension: The nlTGCR algorithm. We now return to the nonlinear problem and ask the question, How can we generalize the algorithms for linear systems of section 3.1 for solving nonlinear equations? We should begin by stating what the desired features of this extension are. First, we would like the algorithms to fall back to TGCR when the problem is linear. Second, we would like a method that can be adapted in such a way as to yield the inexact Newton viewpoint when desired or a multisecant (AA-like) approach when desired. Third, we would like a method that exploits a more accurate linear model than either Newton or a quasi-Newton approach—possibly at the cost of a few extra function evaluations. Finally, we would like the algorithm to be easily adaptable to the very common context in which the function f is "fuzzy," as is the case when dealing with stochastic methods.

In our model, we assume that at step j we have a set of (at most) m current "search" directions $\{p_i\}$ for $j_m \leq i \leq j$ gathered as columns of a matrix P_j, where we recall the notation $j_m \equiv \max\{0, j-m+1\}$. Along with p_i's, we also have a matrix denoted by V_j such that

$$P_j = [p_{j_m}, p_{j_m+1}, \ldots, p_j], \qquad V_j = [v_{j_m}, v_{j_m+1}, \ldots, v_j]. \tag{3.2}$$

Note that this pair of matrices plays the same role as the pair $\mathcal{X}_j$, $\mathcal{F}_j$ defined in (2.13) for AA. We then write the linear model used locally as

$$f(x_j + P_j y) \approx f(x_j) + V_j y. \tag{3.3}$$

While this is again somewhat similar to what was done for AA, we note a very important distinction that may have a significant effect on performance: *In AA the linear model is simply based on the relation $f(x_j - \mathcal{X}_j\theta) \approx f(x_j) - \mathcal{F}_j\theta$, whereas nlTGCR evaluates explicitly the action of $J(x_i)$ on some vector. This evaluation can be quite accurate if desired.* In contrast, the relation $f(x_j - \mathcal{X}_j\theta) \approx f(x_j) - \mathcal{F}_j\theta$ can be quite rough, especially at the beginning of the process where the vectors Δx_j and Δf_j are usually not small.

The projection method will minimize the norm $\|f(x_j) + V_j y\|_2$. This is achieved by determining y in such a way that

$$f(x_j) + V_j y \perp \text{Span}\{V_j\} \;\rightarrow\; (V_j)^T[f(x_j) + V_j y] = 0 \;\rightarrow\; y = V_j^T r_j, \tag{3.4}$$

where it is assumed the v_i's are orthonormal.

Instead of (2.3)–(2.4) of the inexact Newton approach we now have an iteration of the form

$$\text{Find} \qquad y_j = \text{argmin}_y \|f(x_j) + V_j y\|_2, \tag{3.5}$$

$$\text{Set} \qquad x_{j+1} = x_j + P_j y_j. \tag{3.6}$$

A major distinction between this approach and the standard inexact Newton is that the latter exploits approximations to the Jacobian around one point in order to build the next iterate. The iterates generated by the iterative process can be viewed as intermediate points but they rely on a Jacobian $J(x_0)$ calculated at the initial approximation x_0. We will revisit the inexact Newton viewpoint in a later section.

The idea of the nonlinear version of the truncated GCR method is to exploit the directions that are produced by the TGCR algorithm. Note that there is a decoupling between the *update* from the current iterate (lines 4–5 of GCR/TGCR) to a new one and the *construction of the p_i's* in TGCR (lines 7a–7f of TGCR). In essence, the first part just builds a new approximation given a new "search" subspace—while the second adds a new direction to this evolving subspace. This distinction will help us generalize our approach to cases where the objective function or the Jacobian varies as the iteration proceeds.

We now derive our general algorithm from which a few variants will follow. The algorithm is an extension of the TGCR(m) algorithm discussed above—with a few needed changes that reflect the nonlinearity of the problem. The first change is that any residual is now to be replaced by the negative of $f(x)$ so r_0 becomes $r_0 = -f(x_0)$ and line 6 of GCR/TGCR(m) must be replaced by $r_{j+1} = -f(x_{j+1})$. In addition, the matrix A in the products Ar_0 and Ap invoked in line 2 and line 7a, respectively, is the Jacobian of f at the most recent iterate. The most important change is in lines 4–5 of Algorithm 3.1, where according to the above discussion the scalar α_j is to be replaced by the vector y_j that minimized $\|f(x_j) + V_j y\|_2$ over y. The reason for this was explained above.

The resulting nlTGCR(m) algorithm is shown in Algorithm 3.2. It requires two function evaluations per step: one in line 8 and the other in line 10. Alternatively,

Algorithm 3.2 nlTGCR(m).

1: **Input**: $f(x)$, initial x_0.
2: Set $r_0 := -f(x_0)$.
3: Compute $v := J(x_0)r_0$; (Use Frechet)
4: $v_0 := v/\|v\|$, $p_0 := r_0/\|v\|_2$;
5: **for** $j = 0, 1, 2, \ldots,$ **do**
6: $\quad y_j := V_j^T r_j$
7: $\quad x_{j+1} := x_j + P_j y_j$
8: $\quad r_{j+1} := -f(x_{j+1})$
9: $\quad$ Set: $p := r_{j+1}$; and $j_m := \max(0, j - m + 1)$
10: $\quad$ Compute $v = J(x_{j+1})p$ (Use Frechet)
11: $\quad$ **for** $i = j_m : j$ **do**
12: $\quad\quad \beta_{ij} := \langle v, v_i \rangle$
13: $\quad\quad p := p - \beta_{ij} p_i$
14: $\quad\quad v := v - \beta_{ij} v_i$
15: $\quad$ **end for**
16: $\quad p_{j+1} := p/\|v\|_2$; $v_{j+1} := v/\|v\|_2$;
17: **end for**

when constructing the Jacobian is inexpensive, one can compute Jp in line 10 as a matrix-vector product. Clearly, the system $[v_{j_m}, v_{j_m+1}, \ldots, v_{j+1}]$ is orthonormal.

4. Theoretical results. This section discusses connections of the nlTGCR(m) algorithm with inexact and quasi-Newton methods and analyzes its convergence.

4.1. Linearized update version and the connections to inexact Newton methods. We first consider a variant of the algorithm which we call the "linearized update version." We will show that this version is equivalent to inexact Newton methods in which the system is approximately solved with TGCR(m). Two changes are made to Algorithm 3.2 to obtain this linearized update version. First, in line 8 we update the residual by using the linear model, namely, we replace line 8 by

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

The second change is that the Jacobian is not updated in line 10, i.e., $J(x_{j+1})$ in line 10 is kept constant and equal to $J(x_0)$. In other words v is computed as

10a: $\qquad v = J(x_0)p$

The algorithm is stopped when the residual norm r_{j+1} is small enough or the number of steps is exceeded. In addition, *we consider the algorithm merely as a means of providing a search direction* as is often done with inexact Newton methods. In other words the direction $d = x_{last} - x_0$ is provided to another function that will use it in an iterative procedure that includes a line search technique at x_0.

It is easy to see that one iteration defined as a sweep using j substeps of this linear update version is nothing but *an inexact Newton method in which the system* (2.3) *(with $j = 0$) is approximately solved with the TGCR(m) algorithm.* Indeed, in this situation the two main loops (lines 3–8 of Algorithm GCR/TGCR(m) and lines 5–17 of Algorithm 3.2) are identical. Lines 5–17 of Algorithm 3.2 perform k steps of the TGCR(m) algorithm to solve the linear systems $f(x_0) + J(x_0)Py = 0$. Within Algorithm 3.2 the update is written in a progressive form as $x_{j+1} = x_j + \alpha_j p_j$; note that *the right-hand side does not change during the algorithm and is equal to* $r_0 = -f(x_0)$. In effect the last iterate, x_k, is updated from x_0 by adding a vector from the Krylov subspace or equivalently the span of P_k. As a consequence of this observation, it turns out that y_j has only one nonzero component, namely the last one. Indeed, from [22, Thm. 4.1], we see that if we replace A by the Jacobian J at x_0, then

$$\langle r_{j+1}, Jp_i \rangle = 0 \text{ for } i = (j-1)_m, \ldots, j.$$

It was shown in [12] that under mild conditions, the update to the iterate is a descent direction. In addition, the article describes "global convergence strategies" based on line search and trust-region techniques. If we do not apply a global convergence strategy, then the algorithm may have difficulties converging.

Inexact Newton methods [21] are often implemented with residual reduction stopping criteria of the form

$$\|f(x_j) + J(x_j)\delta_j\|_2 \leq \eta_j \|f(x_j)\|_2, \tag{4.1}$$

where $\eta_j \in [0, 1)$ is called the forcing term. This only means that the iterative procedure that is applied when approximately solving the linear system (2.3) exists when the relative residual norm falls below η_j. A number of articles established convergence conditions under conditions based on this framework; see, e.g., [21, 12, 13, 23] among others.

Probably the most significant disadvantage of inexact 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 iterate. After this iterate is computed, all the information gathered at this step, e.g., P_k, 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.

4.2. Nonlinear update version. The "linearized update version" of Algorithm 3.2 discussed above uses a simple linear model to update the residual r_j in which the Jacobian is not updated. Next, we consider the "nonlinear update version" as described in Algorithm 3.2.

Assume a sweep using j substeps of Algorithm 3.2 has been carried out. Our next result will invoke the linear residual

$$\tilde{r}_{j+1} = r_j - V_j y_j, \tag{4.2}$$

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

$$z_{j+1} = \tilde{r}_{j+1} - r_{j+1}. \tag{4.3}$$

We first analyze the magnitude of z_{j+1}. Define

$$w_i = J(x_j)p_i - v_i \text{ for } i = j_m, \ldots, j; \quad \text{and} \quad W_j = [w_{j_m}, \ldots, w_j], \tag{4.4}$$

so that

$$J(x_j)p_i = v_i + w_i. \tag{4.5}$$

Note that the algorithm suggests that $J(x_i)p_i \approx v_i$ but that equality is not satisfied except in the linear case.[3] We also define

$$s_j = f(x_{j+1}) - f(x_j) - J(x_j)(x_{j+1} - x_j). \tag{4.6}$$

Recall from the Taylor series expansion s_j is a second order term relative to $\|x_{j+1} - x_j\|_2$. Then we derive the following bound on the norm of z_{j+1} in the next proposition.

PROPOSITION 4.1. *The difference $\tilde{r}_{j+1} - r_{j+1}$ satisfies the relation*

$$\tilde{r}_{j+1} - r_{j+1} = W_j y_j + s_j = W_j V_j^T r_j + s_j, \tag{4.7}$$

and therefore

$$\|\tilde{r}_{j+1} - r_{j+1}\|_2 \le \|W_j\|_2 \ \|r_j\|_2 + \|s_j\|_2. \tag{4.8}$$

Proof. We rewrite the difference as

$$\begin{aligned} \tilde{r}_{j+1} - r_{j+1} &= (-f(x_j) - V_j y_j) + f(x_{j+1}) \\ &= (f(x_{j+1}) - f(x_j)) - V_j y_j \\ &= J(x_j)P_j y_j + s_j - V_j y_j. \end{aligned} \tag{4.9}$$

[3]We thank Eric de Sturler for catching a statement in an earlier version of this paper related to this observation.

Letting $y_j = [\eta_{j_m}, \eta_{j_m+1}, \cdots]$ we get from (4.5)

$$\begin{aligned} J(x_j)P_j y_j &= \sum \eta_i J(x_j) p_i = \sum \eta_i [v_i + w_i] \\ &= V_j y_j + W_j y_j. \end{aligned} \tag{4.10}$$

The proof follows by combining (4.9) and (4.10). ∎

Proposition 4.1 shows that z_{j+1} is a quantity of the second order: when the process is nearing 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 (4.6). Furthermore, we can prove the following properties of Algorithm 3.2.

PROPOSITION 4.2. *The following properties are satisfied by the vectors produced by Algorithm* 3.2*:*

1. $(\tilde{r}_{j+1}, v_i) = 0$ *for* $j_m \le i \le j$, *i.e.,* $V_j^T \tilde{r}_{j+1} = 0$;
2. $\|\tilde{r}_{j+1}\|_2 = \min_y \| - f(x_j) + V_j y\|_2$;
3. $\langle v_{j+1}, \tilde{r}_{j+1} \rangle = \langle v_{j+1} r_j \rangle$;
4. $y_j = V_j^T r_j = \langle v_j, \tilde{r}_j \rangle e_{m_j} - V_j^T z_j$, *where* $e_{m_j} = [0, 0, \ldots, 1]^T \in \mathbb{R}^{m_j}$.

Proof. Properties 1 and 2 follow from the definition of the algorithm. For property 3 first observe that $V_j = [v_{j_m}, v_{j_m+1}, \ldots, v_{j-1}, v_j]$ and that v_{j+1} is made orthogonal against the m_j vectors v_{j_m} to v_j, so $v_{j+1}^T V_j = 0$ and

$$v_{j+1}^T \tilde{r}_{j+1} = v_{j+1}^T [r_j - V_j y_j] \; = \; v_{j+1}^T r_j.$$

It is convenient to prove property 4 for the index $j+1$ instead of j. We write $V_{j+1}^T r_{j+1} = V_{j+1}^T [\tilde{r}_{j+1} - z_{j+1}]$, where z_{j+1} was defined in (4.3). Recalling properties 1 and 3, we have that $\langle \tilde{r}_{j+1}, v_i \rangle = 0$ for $j_m \le i \le j$—so there is only one nonzero term in the product $V_{j+1}^T \tilde{r}_{j+1}$, namely $v_{j+1}^T \tilde{r}_{j+1}$. This gives the results after adjusting for the change of index. ∎

Property 4 in Proposition 4.2 indicates that when z_j is small, as when the model is *close to being linear* or when it is nearing convergence, then y_j will have small components everywhere except for the last component. As a result it is also possible to consider a slight variant of the algorithm in which y_j is truncated so as to contain only its last entry. We will discuss this variant in detail in section 4.4.

Adaptive update. The nonlinear update version of nlTGCR generally exhibits greater robustness compared to the linearized update version, particularly during the initial phases. In order to leverage the advantages of reduced function evaluations offered by the linearized update version, we introduce an *adaptive update version.* As indicated by property 4 in Proposition 4.2, we employ a mechanism for checking residuals that controls the transition between the two update versions. Let r_j^{nl} and r_j^{lin} represent the nonlinear and linear residuals at iteration j, respectively:

$$r_{j+1}^{nl} = -f(x_{j+1}), \quad r_{j+1}^{lin} = r_j^{nl} - V_j y_j. \tag{4.11}$$

We define the "cosine distance" between the nonlinear and linear residuals as follows:

$$\theta_j := 1 - \frac{(r_j^{nl})^T r_j^{lin}}{\|r_j^{nl}\|_2 \cdot \|r_j^{lin}\|_2}. \tag{4.12}$$

We choose a threshold θ which is 0.01 in this paper. Then, the linearized update version is engaged when $\theta_j < \theta$, while the nonlinear update version is active when

$\theta_j \geq \theta$. Note that, when switching from the linearized version to the nonlinear version, the previous vectors P_j and V_j are reset, initiating a new start for nlTGCR with a beginning point at x_j. This mechanism effectively reduces the overall number of function evaluations without compromising convergence or accuracy.

4.3. Connections to quasi-Newton and multisecant methods. In this section, we show that nlTGCR(m) can be viewed from the alternative angle of a quasi-Newton/multisecant approach. In this viewpoint, the inverse of the Jacobian is approximated progressively. Because it is the inverse Jacobian that is approximated, the method is akin to Broyden's second update method.

First observe that in nlTGCR the update at step j takes the form

$$x_{j+1} = x_j + P_j V_j^T r_j = x_j + P_j V_j^T (-f(x_j)).$$

Thus, we are in effect using a secant-type method. The approximate inverse Jacobian at step j, denoted by G_{j+1} for consistency with common notation, is

$$G_{j+1} = P_j V_j^T. \tag{4.13}$$

If we apply this to v_i we get $G_{j+1} v_i = P_j V_j^T v_i = p_i$ for $j_m \leq i \leq j$. It therefore satisfies the *secant* equation

$$G_{j+1} v_i = p_i \quad \text{for} \quad j_m \leq i \leq j, \tag{4.14}$$

which is a version of (2.10) used in Broyden's second update method. Here, p_i plays the role of Δx_j and v_i plays the role of Δf_j. In addition, the update G_{j+1} satisfies the "no-change" condition:

$$G_{j+1} q = 0 \quad \forall q \perp v_i \quad \text{for} \quad j_m \leq i \leq j. \tag{4.15}$$

The usual no-change condition for secant methods is of the form $(G_{j+1} - G_{j-m})q = 0$ for $q \perp \Delta f_j$, which in our case would be $(G_{j+1} - G_{j-m})q = 0$ for $q \perp v_i$ for $j_m \leq i \leq j$. One can therefore consider that we are updating $G_{j-m} \equiv 0$.

Thus, consider the optimization problem

$$\min\{\|G\|_F \text{ subject to } GV_j = P_j\}, \tag{4.16}$$

which will yield the matrix of the smallest F-norm satisfying the condition (4.14). Not surprisingly this matrix is just G_{j+1}.

PROPOSITION 4.3. *The unique minimizer of problem* (4.16) *is the matrix* G_{j+1} *given by* (4.13).

Proof. The index j is dropped from this proof. Exploiting orthogonal projectors we write G as $G = GVV^T + G(I - VV^T) = PV^T + G(I - VV^T)$ and observe that

$$\begin{aligned} \|G\|_F^2 &= \operatorname{tr}\left([PV^T + G(I - VV^T)][VP^T + (I - VV^T)G^T]\right) \\ &= \operatorname{tr}\left(PP^T\right) + \operatorname{tr}\left(G(I - VV^T)(I - VV^T)G^T\right) \\ &= \|P\|_F^2 + \|G(I - VV^T)\|_F^2. \end{aligned}$$

The right-hand side is minimized when $G(I - VV^T) = 0$, which means when $G = GVV^T$. Recalling the constraint $GV = P$ yields the desired result. ∎

It is also possible to find a link between the method proposed herein and AA reviewed in section 2.3, by unraveling a relation with multisecant methods. Based on Proposition 4.3, we know that

$$G_{j+1}V_j = P_j. \tag{4.17}$$

This is similar to the multisecant condition $G_j\mathcal{F}_j = \mathcal{X}_j$ of (2.19)—see also equation (13) of [61], where $\mathcal{F}_j$ and $\mathcal{X}_j$ are defined in (2.13). In addition, we also have a multisecant version of the no-change condition (4.15). This is similar to a block version of the no-change condition equation (2.12) as represented by equation (15) of [61], which stipulates that

$$(G_j - G_{j-m})q = 0 \quad \forall q \perp \text{Span}\{\mathcal{F}_j\}. \tag{4.18}$$

Strong links can be established with the class of multisecant methods to which AA belongs. Without loss of generality and in an effort to simplify notation we also assume that $j_m = 1$ this time and $\beta = 0$. According to (2.14)–(2.15), the jth iterate becomes simply $x_{j+1} = x_j - \mathcal{X}_j\theta_j$, where θ_j is a vector that minimizes $\|f_j - \mathcal{F}_j\theta\|$. For nlTGCR(m), we have $x_{j+1} = x_j + P_j y_j$, where y_j minimizes $\|f(x_j) + V_j y\|$. So this is identical with (2.14) when $\beta = 0$ in which $P_j \equiv \mathcal{X}_j$, and $\mathcal{F}_j \equiv V_j$:

AA	$\mathcal{X}_j$	$\mathcal{F}_j$	θ_j
nlTGCR	P_j	V_j	$-y_j$

In multisecant methods we set

$$\mathcal{F}_j = [f_1 - f_0, f_2 - f_1, \ldots, f_j - f_{j-1}] \quad \mathcal{X}_j = [x_1 - x_0, x_2 - x_1, \ldots, x_j - x_{j-1}]$$

and, with $G_{j-m} = 0$, the multisecant matrix in (2.18) becomes

$$G_j = \mathcal{X}_j(\mathcal{F}_j^T\mathcal{F}_j)^{-1}\mathcal{F}_j^T. \tag{4.19}$$

We restate the result from [61] that characterizes multisecant methods also known as generalized Broyden techniques [26], for the particular case in which $G_{j-m} \equiv 0$:

- G_j in (4.19) is the only matrix that satisfies both the secant condition (2.19) and the no-change condition (4.18);
- G_j is also the matrix that minimizes $\|G\|_F$ subject to the condition $G\mathcal{F}_j = \mathcal{X}_j$.

Consider now nlTGCR. If we set $\mathcal{F}_j \equiv V_j$ and $\mathcal{X}_j = P_j$ in Anderson, the multisecant matrix G_j in (4.19) simplifies to (4.17)—which also minimizes $\|G\|_F$ under the secant condition $GV_j = P_j$. Therefore 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 the pair of subspaces.

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

$$v_j \approx J(x_j)p_j. \tag{4.20}$$

In the case of nlTGCR(m) this relation is explicitly enforced by a Frechet differentiation (line 10). In the case of AA, we have $v_j = \Delta f_{j-1} = f_j - f_{j-1}$ and the relation exploited is that

$$f_j \approx f_{j-1} + J(x_{j-1})(x_j - x_{j-1}) \rightarrow \Delta f_{j-1} \approx J(x_{j-1})\Delta x_{j-1}. \tag{4.21}$$

However, relation (4.20) in nlTGCR is *more accurate* because we use an additional function evaluation to explicitly obtain an accurate approximation (ideally exact value) for $J(x_j)r_j$ in line 10, which is likely to lead to a smaller error in (4.20). In contrast, when x_j and x_{j-1} are not close, then (4.21) can be a very rough approximation. This is a key difference between the two methods.

4.4. Line search techniques and convergence analysis. In this section, we discuss how to include line search techniques to improve the global convergence of nlTGCR(m). More specifically, if $d_j = P_j y_j$, then line 7 of Algorithm 3.2 will be replaced by $x_{j+1} = x_j + \alpha_j d_j$ with a suitable stepsize α_j.

When nlTGCR(m) is applied to solve a nonlinear system $f(x) = 0$, we define the scalar function $\phi(x) = \frac{1}{2}\|f(x)\|_2^2$. Since $\nabla\phi(x) = J(x)^T f(x)$ and $r_j = -f(x_j)$, the stepsize α_j is chosen to fulfill the Armijo–Goldstein condition [4]:

$$\|f(x_j + \alpha_j d_j)\|_2^2 \le \|r_j\|_2^2 - 2c_1 \cdot \alpha_j \langle J(x_j)^T r_j, d_j\rangle. \tag{4.22}$$

If we approximate the nonlinear residual $-f(x_{j+1})$ with the linearized one $r_{j+1} = r_j - V_j y_j$, we need to replace the left-hand side of the inequality (4.22) by $\|r_j - \alpha_j V_j y_j\|_2^2$. Here d_j is a descent direction for ϕ if $\langle J(x_j)^T r_j, d_j\rangle > 0$.

It is possible to inexpensively check the above condition by first noting that

$$\langle J(x_j)^T r_j, d_j\rangle = \langle r_j, J(x_j) d_j\rangle.$$

Then, using Frechet differentiation we can write

$$\langle J(x_j)^T r_j, d_j\rangle \approx \frac{1}{\epsilon}\langle r_j, f(x_j + \epsilon d_j) - f(x_j)\rangle, \tag{4.23}$$

where ϵ is some small parameter. We already have $f(x_j)$ available and in the context of line search techniques $f(x_j + \lambda_0 d_j)$ is computed as the first step, where λ_0 is some parameter, often set to $\lambda_0 = 1$. If $\|\lambda_0 d_j\|$ is small enough relative to x_j we can get an accurate estimate of $\langle J(x_j)^T r_j, d_j\rangle$ using relation (4.23) with ϵ replaced by λ_0. If not, we may compute an additional function evaluation to obtain $f(x_j + \epsilon d_j)$ for a small ϵ to get the same result with high accuracy. Note that as the process nears convergence $\|d_j\|_2$ becomes small and this is unlikely to be needed.

Often, the direction $d_j = P_j y_j$ is observed to be a descent direction and this can be explained from the result of Proposition 4.2. A few algebraic manipulations lead to the following proposition.

PROPOSITION 4.4. *Let $\phi(x) = \frac{1}{2}\|f(x)\|_2^2$ and let $\tilde{v}_{j_m}, \ldots, \tilde{v}_j$ be the columns of*

$$\tilde{V}_j \equiv J(x_j) P_j. \tag{4.24}$$

Then,

$$\langle \nabla\phi(x_j), d_j\rangle = -\sum_{i=j_m}^{j} \langle v_i, r_j\rangle\langle \tilde{v}_i, r_j\rangle = -r_j^T \tilde{V}_j V_j^T r_j. \tag{4.25}$$

Proof. From property 4 of Proposition 4.2 we have $V_j^T r_j = \langle v_j, \tilde{r}_j\rangle e_{m_j} - V_j^T z_j$ and so

$$\begin{aligned}\langle \nabla\phi(x_j), d_j\rangle &= -\langle J(x_j)^T r_j, P_j[\langle v_j, \tilde{r}_j\rangle e_{m_j} - V_j^T z_j]\rangle \\ &= -\langle v_j, \tilde{r}_j\rangle\langle J(x_j)^T r_j, p_j\rangle + \langle J(x_j)^T r_j, P_j V_j^T z_j\rangle \\ &= -\langle v_j, \tilde{r}_j\rangle\langle r_j, J(x_j)p_j\rangle + \langle r_j, J(x_j)P_j V_j^T z_j\rangle \\ &= -\langle v_j, r_j + z_j\rangle\langle \tilde{v}_j, r_j\rangle + \langle r_j, \tilde{V}_j V_j^T z_j\rangle \\ &= -\langle v_j, r_j\rangle\langle \tilde{v}_j, r_j\rangle - \langle v_j, z_j\rangle\langle \tilde{v}_j, r_j\rangle + \langle r_j, \tilde{V}_j V_j^T z_j\rangle. \qquad (4.26)\end{aligned}$$

We write the last term $\langle r_j, \tilde{V}_j V_j^T z_j\rangle$ in the form

$$\sum_{i=j_m}^{j} r_j^T \tilde{v}_i v_i^T z_j = \sum_{i=j_m}^{j} \langle \tilde{v}_i, r_j\rangle\langle v_i, z_j\rangle.$$

We now note that the last term in the above sum is equal to the middle term in (4.26). The result is that

$$\langle \nabla\phi(x_j), d_j\rangle = -\langle v_j, r_j\rangle\langle \tilde{v}_j, r_j\rangle + \sum_{i=j_m}^{j-1} \langle r_j, \tilde{v}_i v_i^T z_j\rangle. \qquad (4.27)$$

Next observe that for each of the terms in the sum we have

$$v_i^T z_j = v_i^T(\tilde{r}_j - r_j) = -v_i^T r_j \to \sum_{i=j_m}^{j-1} \langle r_j, \tilde{v}_i v_i^T z_j\rangle = -\sum_{i=j_m}^{j-1} \langle \tilde{v}_i, r_j\rangle\langle v_i, r_j\rangle.$$

Substituting this in (4.27) yields the desired result. ∎

We have $v_i \approx J(x_i)p_i$ while $\tilde{v}_i = J(x_j)p_i$. Near convergence, the two vectors will be close enough that the inner products $\langle v_i, r_j\rangle$ and $\langle \tilde{v}_i, r_j\rangle$ will have the same sign, in which case the inner product (4.25) is negative and d_j is a descent direction. In addition, we saw in the proof that $\langle v_i, r_j\rangle = -\langle v_i, z_j\rangle$ for $j_m \le i \le j-1$. Thus, the terms $\langle v_i, r_j\rangle\langle \tilde{v}_i, r_j\rangle$ are likely to be smaller order terms for $i < j$.

Based on the above discussion, we may assume that the direction produced by Algorithm 3.2 is likely to always be a descent direction, but this not guaranteed. However, if needed, this can be inexpensively checked as described earlier at a small additional cost.

In order to ensure global convergence, we also implemented backtracking line search. Specifically, an initial stepsize $\alpha_j^{(0)}$ is defined where the superscript indicates the backtracking steps. We repeatedly set $\alpha_j^{(k+1)} := \tau \cdot \alpha_j^{(k)}$ for a shrinking parameter $\tau \in (0,1)$ and check if the Armijo–Goldstein condition is satisfied or if we have exceeded the maximum allowed number of backtracking steps. Our tests use $\tau = 0.8$. In addition, we found it effective to select the initial stepsize $\alpha_j^{(0)}$ adaptively in order to reduce the number of line search steps. For example, letting $\alpha_0^{(0)} := 1$, we define

$$\alpha_{j+1}^{(0)} := \begin{cases} \min\{1,\, \alpha_j^{(0)}/\tau\} & \text{if line search finishes in 1 step;} \\ \tau \cdot \alpha_j^{(0)} & \text{otherwise.} \end{cases} \qquad (4.28)$$

It is possible to prove that under a few assumptions nlTGCR(m) converges globally. Suppose that we have a line search procedure which at the jth step considers iterates of the form

$$x_{j+1}(t) = x_j + t d_j, \quad \text{with} \quad d_j = P_j y_j = P_j V_j^T r_j. \qquad (4.29)$$

We further assume that the line search is exact, i.e.,
Assumption 1:

$$x_{j+1} = \text{argmin}_{x_{j+1}(t), t\in[0,1]} \|f(x_{j+1}(t))\|.$$

We expand $f(x_j + td_j)$ locally as follows:

$$f(x_j + td_j) = f(x_j) + tJ(x_j)d_j + s_j(t). \tag{4.30}$$

The term $s_j(t)$ is a second order term and we will make the following smoothness assumption:
Assumption 2: There is a constant $K > 0$ such that for each j we have

$$\|s_j(t)\| \le K\|f(x_j)\|t^2. \tag{4.31}$$

In addition, we will assume, without loss of generality, that $K \ge \frac{1}{2}$.

Next, for the term $J(x_j)d_j$ in (4.30), which is equal to $J(x_j)P_jy_j$, it is helpful to exploit the notation (4.4) and observation (4.5). Indeed, these imply that

$$J(x_j)P_jy_j = V_jy_j + W_jy_j. \tag{4.32}$$

It was argued in the discussion following Proposition 4.1 that W_jy_j can be expected to be much smaller in magnitude than r_j and this leads us to our third assumption.
Assumption 3: There is a scalar $0 \le \mu < 1$ such that for all j,

$$\|W_jy_j\| \le \mu\|r_j\| \ . \tag{4.33}$$

Finally, we assume that each linear least-squares problem is solved with a certain relative tolerance.
Assumption 4: At every step j the least-squares solution V_jy_j satisifies

$$\|f(x_j) + V_jy_j\| \le \eta \ \|f(x_j)\|. \tag{4.34}$$

With these the following theorem can be stated.

THEOREM 4.5. *Let Assumptions* 1–4 *be satisfied and assume that the linear least-squares problem in* (4.34) *is solved with a relative tolerance* η *with* $0 < \eta < 1 - \mu$, *i.e.,* η *satisfies*

$$c \equiv 1 - (\eta + \mu) > 0. \tag{4.35}$$

Then, under these assumptions, we have $0 < 1 - c^2/4K < 1$ *and the following inequality is satisfied at each step* j*:*

$$\|f(x_{j+1})\| \le \left[1 - \frac{c^2}{4K}\right] \ \|f(x_j)\|. \tag{4.36}$$

Proof. Recall that

$$f(x_{j+1}(t)) = f(x_j) + tV_jy_j + tW_jy_j + s_j(t), \tag{4.37}$$

which we rewrite as

$$f(x_{j+1}(t)) = t[f(x_j) + V_jy_j] + (1-t)f(x_j) + tW_jy_j + s_j(t). \tag{4.38}$$

Thus, for any t with $0 \le t \le 1$

$$\begin{aligned}\|f(x_{j+1}(t))\| &\le t\|f(x_j) + V_j y_j\| + [(1-t) + \mu t]\|f(x_j)\| + K\|f(x_j)\|t^2 \\ &\le t\eta\|f(x_j)\| + [(1-t) + \mu t]\|f(x_j)\| + K\|f(x_j)\|t^2 \\ &\le [1 - (1 - \eta - \mu)t]\,\|f(x_j)\| + K\|f(x_j)\|t^2. \end{aligned} \tag{4.39}$$

Let us set $\rho_j = \|r_j\|$. From the definition of c and (4.39) we get

$$\min_t \|f(x_{j+1}(t))\| \le \rho_j \times \min_t \left[Kt^2 - ct + 1\right]. \tag{4.40}$$

The minimum with respect to t of the quadratic function in the brackets is reached for $t_{opt} = c/(2K)$ and the minimum value is

$$\min_t \left[Kt^2 - ct + 1\right] = \left[1 - \frac{c^2}{4K}\right]. \tag{4.41}$$

Note that our assumptions imply that $c > 0$ so $t_{opt} > 0$. In addition, $c \le 1$ and so $c/(2K) \le 1/(2K) \le 1$ (since $K \ge 1/2$). Thus, the (exact) line search for $t \in [0, 1]$ will yield x_{j+1}. Relations (4.41) and (4.40) imply that $\rho_{j+1} \le [1 - c^2/(4K)]\rho_j$ where $|1 - c^2/(4K)| < 1$ under the assumptions on c and K. This proves relation (4.36), which establishes convergence under the stated assumptions. ∎

The theorem suggests that the residual norm for each solve must be reduced by a certain minimum amount in order to achieve convergence. For example, if $\mu = 0.1$ (recall that μ is small under the right conditions) and we set $c = 0.1$, then we would need a reduction of at least 0.8, which is similar to the residual norm reduction required in our experiments by default. Also, the four assumptions do not require f to be convex.

4.5. Extension to the stochastic case. Inspired by the great success of nonlinear acceleration methods in accelerating fixed point iterations, it is natural to ask whether they can be applied to accelerate practical applications, such as stochastic optimization problems [62] in deep learning where gradients are approximated by using a random batch of samples in each iteration. We show that the nlTGCR(m) algorithm can be easily generalized to stochastic cases such as minibatching in deep learning. The main difference is that we now build the pair $\{P_j, V_j\}$ with respect to different objective functions $\phi_j(x)$ instead of only one objective function $\phi(x)$. In this case, line 10 of Algorithm 3.2 becomes $v := J_j(x_{j+1})p$ where $J_j(x)$ indicates the Jacobian corresponding to $\phi_j(x)$. We found it important to also add gradient prenormalization to enhance the stability and speed up the training process. That is, the combined gradient of all layers of the entire model has unit 2-norm. Prenormalizing the gradient is useful because the "search" space Span$\{P_j\}$ is invariant to the magnitude of its component vectors p_j. When the batch size is sufficiently large, the "search" direction originating from the stochastic gradient is close to the one generated from the full-batch gradient. Prenormalization can help mitigate the damaging impact of noisy gradients [14]. Given the significance of this topic, we intend to explore this in future work, focusing on the theoretical aspects of the proposed methods [70].

5. Numerical experiments. This section presents a few experiments to compare nlTGCR(m) with existing methods in the literature. We propose an adaptive mechanism that combines the nonlinear update version with the linearized residual computation. This adaptive version is presented in section 5.1.1 and is implemented

in other experiments by default. All experiments were conducted on a workstation equipped with an Intel i7-9700k CPU and an NVIDIA GeForce RTX 3090 GPU. The first three experiments were implemented in MATLAB 2022b, and the baseline methods were based on the implementations by Kasai [40] and Kelley [42]. Deep learning experiments are implemented using PyTorch [1] and run with GPU acceleration.

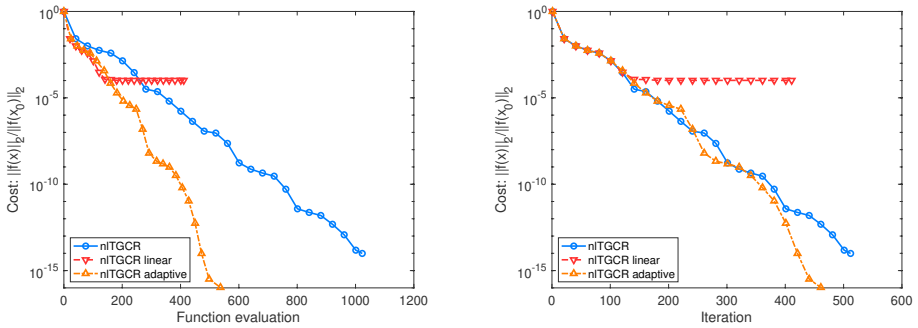
5.1. Bratu problem. We first consider a nonlinear partial differential equation problem, namely the Bratu problem [32] of the following form with $\lambda = 0.5$:

$$\Delta u + \lambda e^u = 0 \quad \text{in} \quad \Omega = (0,1) \times (0,1),$$
$$u(x,y) = 0 \text{ for } (x,y) \in \partial\Omega.$$

This problem is known to be not particularly difficult to solve but our purpose is to illustrate the importance of an accelerator that exploits symmetry. The problem is discretized with centered finite differences [16, 29, 78] using a grid of size 100×100, yielding a nonlinear system of equations $f(s) = 0$ where $f : \mathbb{R}^n \to \mathbb{R}^n$, with $n = 10{,}000$. The corresponding fixed point problem takes the form $g(s) = s + \beta f(s)$.

5.1.1. Superiority of the adaptive update version. The Bratu problem possesses the property of being almost linear despite the presence of an exponential term. This property also appears in numerous applications when nearing convergence. Hence, this problem is suitable to illustrate the difference between the linearized and nonlinear update versions, while also showing the advantage of utilizing the adaptive update version.

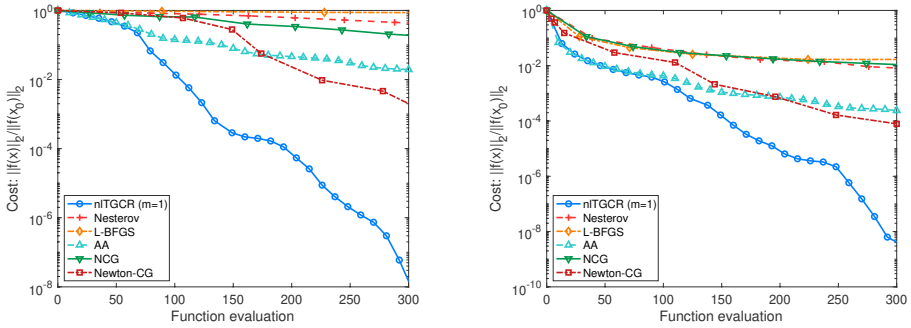
For the experiment in Figure 5.1, the window size is $m = 1$, and the starting point is a vector of all-ones $x_0 = \mathbf{1}$. The cost/objective is the relative residual norm $\|f(x)\|_2/\|f(x_0)\|_2$. We compare all three types of residual update schemes in terms of the number of function evaluations and present results in Figure 5.1(a). It can be observed that the convergence rate of the adaptive update version is close to that of the linearized updated version in the first few iterations. This is because the adaptive update version switches to linear updates at the second iteration and switches back to the nonlinear form at the 110th iteration, where the linear update version stalls. As for the cost (Y-axis) of each iteration, Figure 5.1(b) indicates that all three versions decrease almost in the same way per iteration before the onset of stagnation. The adaptive update version performs as well as the nonlinear update version afterward.



(a) Number of function evaluations vs. relative residual norm.

(b) Number of iterations vs. relative residual norm.

FIG. 5.1. *Comparison between the standard, linearized update, and adaptive update versions of nlTGCR(m) with $m = 1$ on the Bratu problem. Each marker represents* 20 *iterations.*



(a) Starting point $x_0 = \mathbf{0}$. (b) Starting point $x_0 = \mathbf{1}$.

FIG. 5.2. *Number of function evaluations versus relative residual norm on the Bratu problem with different starting points. Each marker represents* 10 *iterations, except Newton-CG, where each marker represents* 1 *outer loop step.*

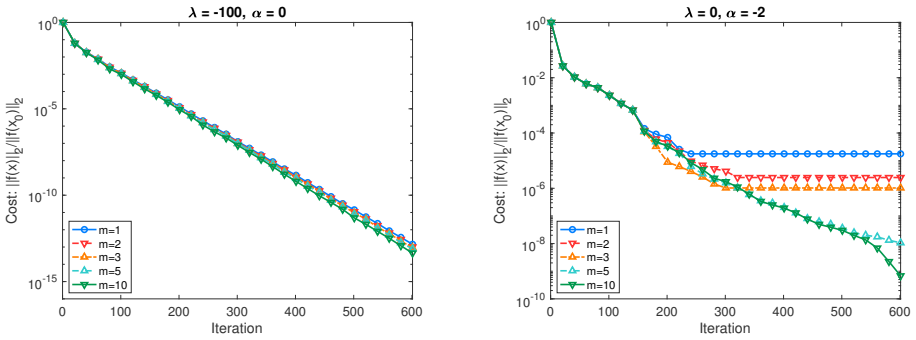
Sections 5.1.2 and 5.2 report on more experiments with the adaptive version of nlTGCR(m). In sections 5.3 to 5.5 we utilize the standard (nonlinear) update version of nlTGCR(m) for deep learning tasks, as the proposed residual check is not applicable in a stochastic context.

5.1.2. Exploiting symmetry. We now investigate whether nlTGCR(m) takes advantage of short-term recurrences when the Jacobian is symmetric. Recall that in the linear case, GCR is mathematically equivalent to the conjugate residual algorithm when the matrix is symmetric. In this situation a window size $m = 1$ is optimal [63]. We tested nlTGCR(m) along with baselines including Nesterov's accelerated gradient [54], L-BFGS [47], AA, nonlinear conjugate gradient (NCG) of Fletcher–Reeves type [28], and Newton-CG [21]. Results are presented in Figures 5.2(a) and 5.2(b). We analyzed the convergence in terms of the number of function evaluations rather than the number of iterations because the backtracking line search is implemented for all methods considered except AA by default. We present the results of the first 300 function evaluations for all methods. The window size for nlTGCR(m) is $m = 1$, while for L-BFGS and AA, it is $m = 10$. The mixing parameter for AA is set to $\beta = 0.1$ as suggested in [9]. For Newton-CG method, the maximum number of steps in the inner CG solve is 50. This inner loop can be terminated early if

$$\|r_k\| \leq \eta \|f(x_0)\|. \tag{5.1}$$

We choose the forcing term $\eta = 0.9$ and adjust it via the Eisenstat–Walker method [24].

The Jacobian for the Bratu problem $\nabla f(s)$ is SPD, making nlTGCR(m) with $m = 1$ a highly efficient method. Other methods that do not take advantage of this symmetry require a larger number of vectors to achieve comparable performance. We tested the methods with two different initial guesses. The first, used in the experiments in Figure 5.2(a), is $x_0 = \mathbf{0}$, which is somewhat close to the global optimum. The second initial guess, used in the experiments in Figure 5.2(b), is the vector of all ones $x_0 = \mathbf{1}$. In both cases, we set the window size of L-BFGS and AA to 10, which means that 20 vectors in addition to x_j and r_j need to be stored. In contrast, nlTGCR(m = 1) only requires 2 extra vectors. In this problem, nlTGCR(m = 1), Nesterov, NCG, and Newton-CG are memory-efficient in terms of the number of vectors required to store information from previous steps. Among these competitive methods, nlTGCR($m = 1$) performs best—suggesting that nlTGCR(m) benefits from symmetry.



(a) Near-linear symmetric Jacobian. (b) Linear asymmetric Jacobian.

FIG. 5.3. *Comparison of nlTGCR(m) with* $m = 1, 2, 3, 5, 10$ *on the modified Bratu problem. Each marker represents* 20 *iterations.*

To further clarify our claim, we modify the Bratu problem

$$\Delta u + \alpha u_x + \lambda e^u = 0$$

by introducing asymmetry to the Jacobian via the term αu_x. We employ the standard nlTGCR method with no restarts, and we consider various table sizes $m = 1, 2, 3, 5,$ 10 and initiate the process using an all-one vector $\mathbf{1}$.

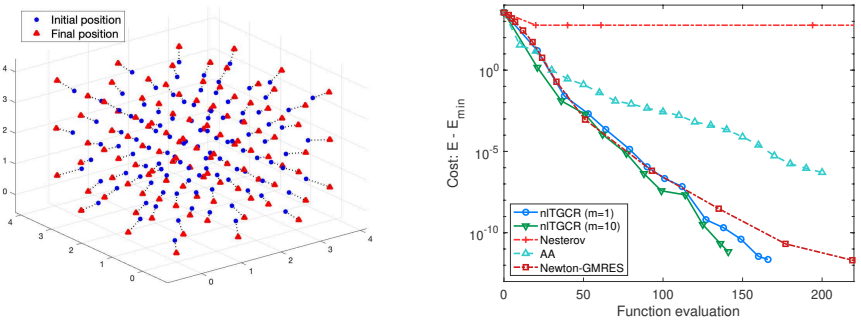
In Figure 5.3(a), we set $\lambda = -100$ and $\alpha = 0$, resulting in a nonlinear symmetric Jacobian. The overlap of all lines confirms the observation that nlTGCR benefits from the symmetry of the Jacobian. Conversely, in Figure 5.3(b) where $\lambda = 0$ and $\alpha = -2$, the Jacobian is linear and asymmetric. The selection of table size m profoundly impacts the performance. Hence, nlTGCR exploits the symmetry to establish short-term recurrence and improve convergence, while the nonlinearity of the problem mainly affects the adaptive update mechanism.

5.2. Molecular optimization with Lennard-Jones potential. The second experiment focuses on the molecular optimization with the Lennard-Jones (LJ) potential, which is a geometry optimization problem. The goal is to find atom positions that minimize total potential energy as described by the LJ potential:[4]

$$E = \sum_{i=1}^{N} \sum_{j=1}^{i-1} 4 \times \left[\frac{1}{\|y_i - y_j\|^{12}} - \frac{1}{\|y_i - y_j\|^6} \right]. \tag{5.2}$$

In the above expression each y_i is a three-dimensional vector whose components are the coordinates of the location of atom i. We 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—it is just a local minimum around the initial configuration (see, e.g., Figure 5.4(a)). In this particular example, we simulate an Argon cluster by taking the initial position of the atoms to consist of a perturbed initial face-centered-cubic (FCC) structure [53]. We took 3 cells per direction—resulting in 27 unit cells. FCC cells include 4 atoms each and so we end up with a total of 108 atoms. The problem is rather hard due to the high powers in the potential. In this situation backtracking or some form of line search is essential.

[4]We benefited from Stefan Goedecker's course site at Basel University.



(a) Geometrical configurations. (b) Convergence of various methods.

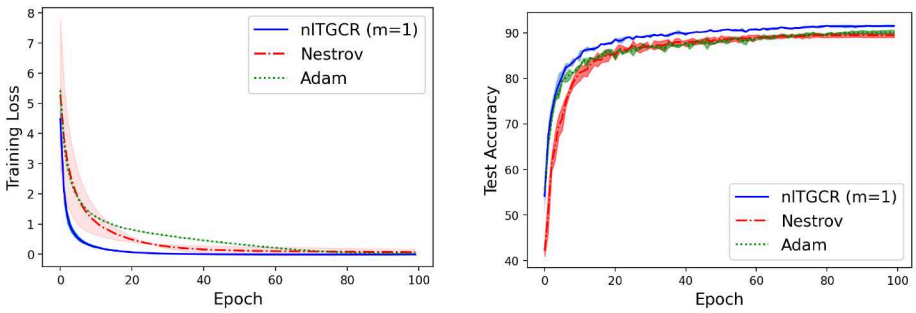
FIG. 5.4. (a) *Initial and final configurations of* 108 *atoms with the Argon cluster experiment.* (b) *Number of function evaluations versus shifted potential norm on the LJ problem. Each marker represents* 10 *iterations for all methods except Newton-GMRES, where each marker represents* 1 *outer loop step.*

In this experiment, we set $f = \nabla E$. Each iterate in nlTGCR(m) is a vectorized array of coordinates of all atoms put together. So, it is a flat vector of length $3 \times 108 = 324$. We present the results of the first 220 function evaluations for nlTGCR(m), AA, Nesterov, and Newton-GMRES in Figure 5.4(b). The reason for excluding L-BFGS, NCG, and Newton-CG is that the Jacobian/Hessian of the LJ problem is indefinite at some x_j, which can lead to a nondescending update direction. The window sizes for nlTGCR(m) are $m = 1$ and $m = 10$, while for AA it is $m = 10$ and for Newton-GMRES it is $m = 20$. This is because nlTGCR(m) and AA require storing twice as many additional vectors as the window size to generate the searching subspace, while Newton-GMRES does not. For each inner GMRES solve, GMRES is allowed to run up to 40 steps and utilizes a forcing term $\eta = 0.9$ to terminate the inner loop. Moreover, AA does not converge unless the underlying fixed point iteration $s_{j+1} = s_j + \beta f(s_j)$ is carefully chosen. In this experiment, we select $\beta = 10^{-3}$. The cost (Y-axis) represents the shifted potential $E - E_{min}$, where E_{min} is the minimal potential achieved by all methods, approximately equal to -579.4638.

In Figure 5.4(b), we observe that nlTGCR($m = 10$) converges the fastest. The convergence of nlTGCR($m = 1$) and Newton-GMRES with a subspace dimension of 20 is close and slightly slower than nlTGCR($m = 10$). One observed phenomenon worth mentioning is the quick termination of the inner loop of Newton-GMRES during the first few iterations. Newton-GMRES moves quickly to the next Jacobian at the beginning and focuses on a single Jacobian when nearing convergence. This behavior is made possible by the use of the Eisenstat–Walker technique, which accounts for the fast convergence of Newton-GMRES. However, without this early stopping mechanism, Newton-GMRES will fail to converge. In contrast, nlTGCR(m) and AA do not exclusively rely on one Jacobian at each iteration.

5.3. Image classification using ResNet. We now test the usefulness of nlTGCR(m) for deep learning applications by first comparing it with two commonly used optimization algorithms, Adam [43] and momentum. We report the training mean squared error (MSE) and test accuracy on the CIFAR10 dataset [45] using ResNet [37]. We employed a ResNet 18 architecture from PyTorch.[5] The window size

[5]https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py.



(a) Training Loss (b) Test Accuracy

FIG. 5.5. *Image classification on CIFAR*10 *using ResNet (averaged over* 5 *independent runs). nlTGCR(*$m = 1$*), Adam, and momentum achieved a test accuracy of* 91.56%, 90.13%, 89.53%, *respectively.*

m is 1 for nlTGCR(m) because we found a large window size did not help improve the convergence too much in our preliminary experiments. The hyperparameters of the baseline methods are set to be the same as suggested in [37], i.e., the learning rate is 0.001 and 0.1 for Adam and momentum, respectively. Figure 5.5(a) depicts the training loss over time for each optimization algorithm, and Figure 5.5(b) shows the corresponding test accuracy. As can be seen, nlTGCR($m = 1$) achieved the best performance in terms of both convergence speed and accuracy. The experimental results revealed that nlTGCR outperformed the baseline methods by a significant margin. It is worth noting that nlTGCR converges to a loss of 0 for this problem, which empirically verifies the theoretically established global convergence properties of the method. This suggests that nlTGCR($m = 1$) may lead to an interesting alternative to the state-of-the-art optimization methods in deep learning.

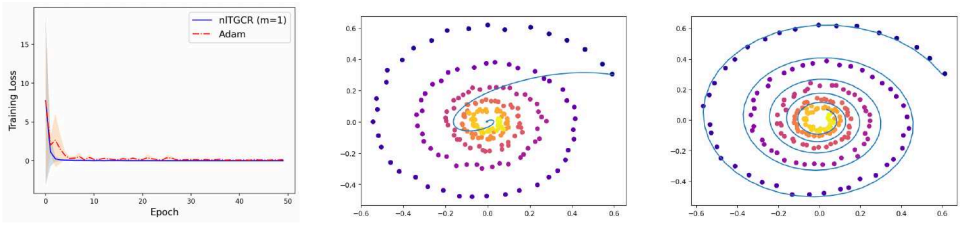
5.4. Learning dynamic using neural-ODE solver. We conducted experiments using a neural-ODE solver [17] to learn underlying dynamic ODEs from sampled data. In our work, we focused on studying the spiral function

$$\frac{dz}{dt} = \begin{bmatrix} -0.1 & -1.0 \\ 1.0 & -0.1 \end{bmatrix} z,$$

which is a challenging dynamic to fit and is often used as a benchmark for testing the effectiveness of machine learning algorithms. We generated the training data by randomly sampling points from the spiral and adding small amounts of Gaussian noise. The goal was to train the model to generate data-like trajectories.

However, training such a model is computationally expensive. Therefore, we introduced nlTGCR(m) to recover the spiral function quickly and accurately, with the potential for better generalization to other functions. Our experiments involved training and evaluating a neural-ODE model on the sampled dataset for the spiral function compared with Adam and momentum. After a grid search, we set the learning rate as 0.01 for Adam and window size $m = 1$ for nlTGCR(m). We reported the MSE training loss and visualized the model's ability to recover the dynamic in Figure 5.6. We did not visualize the momentum results as it took over 100 epochs to converge.

Figure 5.6(a) shows that nlTGCR(m = 1) converges faster and more stably than Adam. Additionally, Figures 5.6(b) and 5.6(c) demonstrate that nlTGCR(m = 1) is capable of generating data-like trajectories in just 15 epochs, whereas Adam struggles



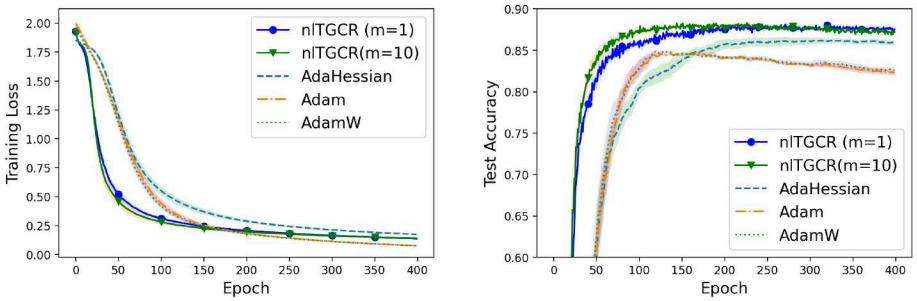
(a) Training Loss (b) Adam after 50 epochs (c) nlTGCR(m=1) after 15 epochs

FIG. 5.6. *Learning true dynamic using neural ODE. It can be observed that nlTGCR($m = 1$) converges much faster and can recover the true dynamic up to* $5\times$ *faster than Adam.*

to converge even after 50 epochs. This experiment demonstrates the superiority of nlTGCR($m = 1$) for this interesting application, relative to commonly used optimizers such as Adam and momentum.

5.5. Node classification using graph convolutional networks. Finally, we explore the effectiveness of nlTGCR(m) in deep learning applications by implementing graph convolutional networks (GCNs) [44]. We use the commonly used Cora dataset [51], which contains 2708 scientific publications on one of 7 topics. Each publication is described by a binary vector of 1433 unique words indicating the absence or presence in the dictionary. This network consists of 5429 links representing the citation. The objective is a node classification via words and links. The neural network has one GCN layer and one dropout layer of rate 0.5. We adopted the GCN implementation directly from PyTorch-Geometric [2].

We set $m = 1$ and $m = 10$ in nlTGCR(m) and compare it with Adam [43], AdamW [48], and AdaHessian [79] with learning rates $lr = 0.01, 0.01, 0.003$, respectively, after a grid search. We present results of the training loss (cross entropy) and test accuracy in Figures 5.7(a) and 5.7(b). Note that a lower loss function does not necessarily mean a better solution or convergence because of the bias and variance trade-off. Although Adam achieved a lower training loss, it is not considered as a better solution because its poor generalization capability on unseen datasets. We can observe that nlTGCR($m = 1$) shows the best performance in this task. Specifically, it is not necessary to use a large window size since there is no significant difference between



(a) Training Loss (b) Test Accuracy

FIG. 5.7. *GCN on Cora. nlTGCR($m = 1$) achieved a test accuracy of* 88.13%, *which is* 4.02% *higher than the second best baseline AdamW.*

$m = 1$ and $m = 10$. This experiment shows again that nlTGCR($m = 1$) can be adapted to the solution of deep learning problems.

6. Concluding remarks. The initial goal of this study was primarily to develop Anderson-like methods that can take advantage of the symmetry of the Hessian in optimization problems. What we hope to have conveyed to the reader is that, by a careful extension of linear iterative schemes, one can develop a whole class of methods, of which nlTGCR(m) is but one member, that can be quite effective, possibly more so than many of the existing techniques in some situations. We are cautiously encouraged by the results obtained for deep learning problems, although much work remains to be done to adapt and further test nlTGCR(m) for the stochastic context. In fact, our immediate research plan is precisely to perform such an in-depth study that focuses on deep learning applications.

Appendix A. More on GCR for solving linear systems. A number of results on the GCR algorithm for linear systems are known but their statements or proofs are not readily available from a single source. For example, it is intuitive, and well-known, that when A is Hermitian then GCR will simplify to its CG-like algorithm known as the conjugate residual algorithm, but a proof cannot be easily found. This section summarizes the most important of these results with an emphasis on a unified presentation that exploits a matrix formalism.

A.1. Conjugacy and orthogonality relations. We start with Algorithm 3.1, where we assume no truncation ($m = \infty$). It is convenient to use a matrix formalism for the purpose of unraveling some relations and so we start by defining

$$R_k = [r_0, r_1, \ldots, r_k]. \tag{A.1}$$

The relation in line 7 of the algorithm can be rewritten in matrix form as

$$R_k = P_k B^{(k)}, \tag{A.2}$$

where $B^{(k)}$ is an upper triangular matrix of size $(k+1) \times (k+1)$ defined as follows:

$$B^{(k)} \in \mathbb{R}^{(k+1)\times(k+1)}, \qquad B^{(k)}_{ij} = \begin{cases} 0 & \text{if} \quad i > j, \\ 1 & \text{if} \quad i = j \\ \beta_{(i-1),(j-1)} & \text{if} \quad i < j. \end{cases} \tag{A.3}$$

Similarly, note that the relation from line 6 of the algorithm can be recast as

$$AP_k = R_{k+1} \underline{H}^{(k)}, \tag{A.4}$$

where $\underline{H}^{(k)}$ is a $(k+2) \times (k+1)$ lower bidiagonal matrix with

$$\underline{H}^{(k)}_{ij} = \begin{cases} \frac{1}{\alpha_{j-1}} & \text{if} \quad i = j, \\ \frac{-1}{\alpha_{j-1}} & \text{if} \quad i = j+1, \\ 0 & \text{if} \quad i < j \text{ or } i > j+1. \end{cases} \tag{A.5}$$

PROPOSITION A.1 (Eisenstat, Elman, and Schultz [22]). *The residual vectors produced by (full) GCR are semiconjugate.*

Proof. Semiconjugacy means that each r_j is orthogonal to $Ar_0, Ar_1, \ldots Ar_{j-1}$, so we need to show that $R_k^T A R_k$ is upper triangular. We know that each r_{j+1} is orthogonal to Ap_i for $i = 1, \ldots, j$, a relation we write as

$$R_k^T A P_k = U_k, \tag{A.6}$$

where U_k is some upper triangular matrix. Then, from (A.2) we have $AR_k = AP_k B^{(k)}$ and therefore

$$R_k^T A R_k = R_k^T A P_k B^{(k)} = U_k B^{(k)}, \tag{A.7}$$

which is upper triangular as desired. ∎

We get an immediate consequence of this result for the case when A is Hermitian.

COROLLARY A.2. *When A is symmetric real (Hermitian complex), then the directions $\{r_j\}$ are A-conjugate.*

Proof. Indeed, when A is symmetric real the matrix $R_k^T A R_k = U_k B^{(k)}$ is also symmetric and since it is upper triangular it must be diagonal, which shows that the r_j's are A-conjugate. ∎

In this situation, we can write

$$R_k^T A R_k = D_k, \tag{A.8}$$

where D_k is a $(k+1) \times (k+1)$ diagonal matrix. The next result shows that the algorithm simplifies when A is symmetric. Specifically, the scalars β_{ij} needed in the orthgonalization in line 7 are all zero except β_{jj}.

PROPOSITION A.3. *When A is symmetric real, then the matrix $(AR_k)^T(AP_k)$ is lower bidiagonal.*

Proof. As a result of (A.4) the matrix $(AR_k)^T(AP_k)$ is

$$(AR_k)^T(AP_k) = (AR_k)^T R_{k+1} \underline{H}^{(k)}.$$

Observe that $(AR_k)^T R_{k+1} = [D_k, 0_{(k+1)\times 1}]$ and the product $[D_k, 0_{(k+1)\times 1}]\underline{H}^{(k)}$ yields the $(k+1) \times (k+1)$ matrix obtained from $\underline{H}^{(k)}$ but deleting its last row which we denote by $H^{(k)}$. Therefore,

$$(AR_k)^T(AP_k) = D_k H^{(k)}$$

is a $(k+1) \times (k+1)$ bidiagonal matrix. ∎

A.2. Breakdown of GCR. Next we examine the conditions under which the full GCR breaks down.

PROPOSITION A.4. *When A is nonsingular, the only way in which (full) GCR breaks down is when it produces an exact solution. In other words, its only possible breakdown is the so-called lucky breakdown.*

Proof. The algorithm breaks down only if Ap_{j+1} produced in line 7 is zero, i.e., when $p_{j+1} == 0$ since A is nonsingular. In this situation $r_{j+1} = \sum_{i=0}^{j} \beta_{ij} p_i$. It can be easily seen that each p_i is of the form $p_i = \mu_i(A) r_0$ where μ_i is a polynomial of degree i. Similarly $r_{j+1} = \rho_{j+1}(A) r_0$ in which ρ_{j+1} is a polynomial of degree (exactly) $j+1$. Therefore, the polynomial $\pi_{j+1}(t) \equiv \rho_{j+1}(t) - \sum_{i=0}^{j} \beta_{ij} \mu_i(t)$ is a polynomial of degree

exactly $j+1$ such that $\pi_{j+1}(A)r_0 = 0$. Thus, the degree of the *minimal polynomial* for r_0 is $j+1$ and we are in the standard situation of a lucky breakdown. Indeed, since the algorithm produces a solution x_{j+1} that minimizes the residual norm, and because the degree of the minimal polynomial for r_0 is equal to $j+1$, we must have $r_{j+1} = 0$. ∎

Note that the proposition does not state anything about convergence. The iterates that are computed will have a residual norm that is nonincreasing but the iterates may stagnate. Convergence can be shown in the case when A is positive definite, i.e., when its symmetric part is SPD.

In addition, the proof of this result requires that the solution that is produced has a minimal residual, which is not the case for the truncated version. Thus, in the truncated version, we may well have a situation where $p_{j+1} = 0$ but the solution x_{j+1} is not exact. If we had $p_{j+1} = 0$ it would mean that $\eta_{j+1}(A)r_0 = 0$, i.e., the minimal polynomial for r_0 is again of degree exactly $j+1$. This is an unlikely event that we may term "unlucky breakdown." However, in practice, the more common situation that can take place is to get a vector p_{j+1} with a small norm.

Suppose now that A is positive definite, i.e., that its symmetric part is SPD. Let us assume that $p_k = 0$ but $r_k \neq 0$—which represents the scenario of an "unlucky breakdown" at step k. Then since the last column of P_k is a zero vector the last column of the matrix U_k in (A.6) is also zero. This in turn would imply that the last *row* of the product $U_k B^{(k)}$ in (A.7) is zero. However, this can't happen because according to (A.7) it is equal to the last row of the matrix $R_k^T A R_k$ where A is positive definite. Thus, the "unlucky breakdown" scenario invoked above is possible only when A is indefinite.

A.3. Properties of the induced approximate inverse. When $x_0 = 0$ the approximate solution obtained at the end of the algorithm is of the form $x_{k+1} = P_k V_k^T b$. We say that the algorithm induces the approximation $B_k = P_k V_k^T$ to the inverse of A. Note that even in the case when A is symmetric, B_k is not symmetric in general. However, B_k obeys a few easy-to-prove properties stated next.

PROPOSITION A.5. *Let $\mathcal{L}_k = Span(V_k)$ and let $\pi = V_k V_k^T$ be the orthogonal projector onto $\mathcal{L}_k$. The (full) GCR algorithm induces the approximate inverse $B_k = P_k V_k^T$ which satisfies the following properties:*

1. *$B_k = A^{-1}\pi$.*
2. *B_k inverts A exactly in $\mathcal{L}_k$, i.e., $B_k x = A^{-1}x$ for $x \in \mathcal{L}_k$. Equivalently, $B_k \pi = A^{-1}\pi$.*
3. *AB_k equals the orthogonal projector π.*
4. *When A is symmetric then B_k is self-adjoint when restricted to $\mathcal{L}_k$.*
5. *$B_k A x = x$ for any $x \in Span\{P_k\}$, i.e., B_k inverts A exactly from the left when A is restricted to the range of P_k.*
6. *$B_k A$ is the projector onto $Span\{P_k\}$ and orthogonally to $A^T\mathcal{L}_k$.*

Proof. (1) The first property follows from the relation $AP_k = V_k$ and the definition of B_k.

(2) To prove the second property we write a member of $\mathrm{Span}(V_k)$ as $x = V_k y$. Then from the previous property we have

$$B_k x = A^{-1} V_k V_k^T V_k y = A^{-1} V_k y = A^{-1} x.$$

(3) $AB_k = AP_k V_k^T = V_k V_k^T = \pi$.

(4) The self-adjointness of B_k in $\mathcal{L}_k$ is a consequence of (2). It can also be readily verified as follows. For any vectors $x, y \in \mathcal{L}_k$ write

$$(B_k x, y) = (A^{-1}\pi x, y) = (A^{-1}x, y) = (x, A^{-1}y) = (x, A^{-1}\pi y) = (x, B_k y).$$

(5) Let a member of $\mathrm{Span}(P_k)$ be written as $x = P_k y$ and apply $B_k A$ to x:

$$B_k A x = B_k A P_k y = B_k V_k y = A^{-1} V_k V_k^T V_k y = A^{-1} V_k y = P_k y = x.$$

Therefore, $B_k A$ leaves vectors of $\mathrm{Span}(P_k)$ unchanged.

(6) Because $B_k A$ leaves vectors of $\mathrm{Span}(P_k)$ unchanged it is a projector, call it π_O (for oblique), with $\mathrm{Ran}(\pi_O) = \mathrm{Span}\{P_k\}$. We now need to show that $(u - \pi_O u) \perp A^T \mathcal{L}_k$ for any u. Since $A^T V_k$ is a basis for $A^T \mathcal{L}_k$, this is equivalent to showing that $(A^T V_k)^T (u - \pi_O u)) = 0$ for any u. We have for any given vector $u \in \mathbb{R}^n$

$$\begin{aligned}(A^T V_k)^T (u - \pi_O u) &= V_k^T (Au - A P_k V_k^T Au) = V_k^T (Au - V_k V_k^T Au) \\ &= V_k^T (I - V_k V_k^T) Au = 0.\end{aligned}$$

∎

Thus (4) and (6) indicate that while AB_k is an orthogonal projector, $B_k A$ is an oblique projector. Although (4) is an obvious consequence of (3), it is interesting to note that it is rather similar to relations obtained in the context of Moore–Penrose pseudoinverses.

Appendix B. Convergence analysis. We provide an alternative analysis of the global convergence of nlTGCR based on the backtracking line search strategy. We will make the following assumptions:

Assumption A:

(B.1) The set $S = \{x : \phi(x) \le \phi(x_0)\}$ is bounded.

Assumption B:

(B.2) $\nabla\phi$ is L-Lipschitz, $\|\nabla\phi(x) - \nabla\phi(y)\| \le L\|x - y\|$, $0 < L < \infty$.

Assumption C: There exists a $\gamma > 0$ such that d_j produced by Algorithm 3.2 satisfies

$$-\frac{d_j^\top \nabla\phi(x_j)}{\|d_j\|\|\nabla\phi(x_j)\|} \ge \gamma > 0 \ \forall j. \tag{B.3}$$

Assumption D: There exist two positive constants μ and Θ such that

$$\|d_j\| \ge \mu\|\nabla\phi(x_j)\|, \quad \|d_j\| \le \Theta \ \forall j. \tag{B.4}$$

THEOREM B.1. *For any scalar function $\phi(x)$, consider the iterates $x_{j+1} = x_j + \alpha_j d_j$ with descent directions d_j produced by Algorithm* 3.2 *and stepsizes α_j produced by backtracking line search* (4.22). *Suppose Assumptions* A–D *also hold; then*

$$\lim_{j\to\infty} \|\nabla\phi(x_j)\| = 0. \tag{B.5}$$

This theorem is a well-known result in optimization and it can be found in various sources, including [31]. We omit its proof.

Acknowledgments. We would like to thank the editor and the two reviewers for their insightful suggestions that have significantly enhanced the quality of our paper.

REFERENCES

[1] *PyTorch*, https://www.pytorch.org.
[2] *PyTorch-Geometric*, https://www.pyg.org/.
[3] D. G. Anderson, *Iterative procedures for non-linear integral equations*, J. ACM, 12 (1965), pp. 547–560.
[4] L. Armijo, *Minimization of functions having Lipschitz continuous first partial derivatives*, Pacific J. Math., 16 (1966), pp. 1–3.
[5] O. Axelsson, *Conjugate gradient type-methods for unsymmetric and inconsistent systems of linear equations*, Linear Algebra Appl., 29 (1980), pp. 1–16.
[6] W. Bian, X. Chen, and C. T. Kelley, *Anderson acceleration for a class of nonsmooth fixed-point problems*, SIAM J. Sci. Comput., 43 (2021), pp. S1–S20, https://doi.org/10.1137/20M132938X.
[7] N. Boutet, R. Haelterman, and J. Degroote, *Secant update version of quasi-Newton PSB with weighted multisecant equations*, Comput. Optim. Appl., 75 (2020), pp. 441–466, https://doi.org/10.1007/s10589-019-00164-z.
[8] N. Boutet, R. Haelterman, and J. Degroote, *Secant update generalized version of PSB: A new approach*, Comput. Optim. Appl., 78 (2021), pp. 953–982, https://doi.org/10.1007/s10589-020-00256-1.
[9] C. Brezinski, S. Cipolla, M. Redivo-Zaglia, and Y. Saad, *Shanks and Anderson-type acceleration techniques for systems of nonlinear equations*, IMA J. Numer. Anal., 42 (2022), pp. 3058–3093.
[10] C. Brezinski and M. Redivo-Zaglia, *The simplified topological ε-algorithms for accelerating sequences in a vector space*, SIAM J. Sci. Comput., 36 (2014), pp. A2227–A2247, https://doi.org/10.1137/140957044.
[11] C. Brezinski, M. Redivo-Zaglia, and Y. Saad, *Shanks sequence transformations and Anderson acceleration*, SIAM Rev., 60 (2018), pp. 646–669, https://doi.org/10.1137/17M1120725.
[12] P. N. Brown and Y. Saad, *Hybrid Krylov methods for nonlinear systems of equations*, SIAM J. Sci. Stat. Comput., 11 (1990), pp. 450–481.
[13] P. N. Brown and Y. Saad, *Convergence theory of nonlinear Newton–Krylov algorithms*, SIAM J. Optim., 4 (1994), pp. 297–330.
[14] A. Cabana and L. F. Lago-Fernández, *Backward Gradient Normalization in Deep Neural Networks*, https://arxiv.org/abs/2106.09475, 2021.
[15] S. Cabay and L. W. Jackson, *A polynomial extrapolation method for finding limits and antilimits of vector sequences*, SIAM J. Numer. Anal., 13 (1976), pp. 734–752, https://doi.org/10.1137/0713060.
[16] M. H. Chaudhry, *Open-Channel Flow*, Springer, New York, 2008.
[17] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, *Neural Ordinary Differential Equations*, https://arxiv.org/abs/1806.07366, 2019.
[18] M. Chupin, M.-S. Dupuy, G. Legendre, and E. Séré, *Convergence analysis of adaptive DIIS algorithms with application to electronic ground state calculations*, ESAIM Math. Model. Numer. Anal., 55 (2021), pp. 2785–2825.
[19] A. d'Aspremont, D. Scieur, and A. Taylor, *Acceleration methods*, Found. Trends Optim., 5 (2021), pp. 1–245, https://doi.org/10.1561/2400000036.
[20] J. Degroote, K.-J. Bathe, and J. Vierendeels, *Performance of a new partitioned procedure versus a monolithic procedure in fluid-structure interaction*, Comput. & Struct., 87 (2009), pp. 798–801.
[21] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal., 18 (1982), pp. 400–408.
[22] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 20 (1983), pp. 345–357.
[23] S. C. Eisenstat and H. F. Walker, *Globally convergent inexact Newton methods*, SIAM J. Optim., 4 (1994), pp. 393–422.
[24] S. C. Eisenstat and H. F. Walker, *Choosing the forcing terms in an inexact Newton method*, SIAM J. Sci. Comput., 17 (1996), pp. 16–32, https://doi.org/10.1137/0917003.
[25] C. Evans, S. Pollock, L. G. Rebholz, and M. Xiao, *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 (2020), pp. 788–810, https://doi.org/10.1137/19M1245384.

[26] V. Eyert, *A comparative study on methods for convergence acceleration of iterative vector sequences*, J. Comput. Phys., 124 (1996), pp. 271–285.

[27] J.-L. Fattebert, *Accelerated block preconditioned gradient method for large scale wave functions calculations in density functional theory*, J. Comput. Phys., 229 (2010), pp. 441–452, https://doi.org/10.1016/j.jcp.2009.09.035.

[28] R. Fletcher and C. M. Reeves, *Function minimization by conjugate gradients*, Comput. J., 7 (1964), pp. 149–154.

[29] G. B. Folland, *Introduction to Partial Differential Equations*, Math. Notes 102, Princeton University Press, Princeton, NJ, 1995.

[30] M. Geist and B. Scherrer, *Anderson Acceleration for Reinforcement Learning*, https://arxiv.org/abs/1809.09501, 2018.

[31] I. Griva, S. G. Nash, and A. Sofer, *Linear and Nonlinear Optimization*, SIAM, Philadelphia, 2009.

[32] M. Hajipour, A. Jajarmi, and D. Baleanu, *On the accurate discretization of a highly nonlinear boundary value problem*, Numer. Algorithms, 79 (2018), pp. 679–695.

[33] H. He, Y. Xi, and J. C. Ho, *Fast and accurate tensor decomposition without a high performance computing machine*, in Proceedings of the IEEE International Conference on Big Data (Big Data), IEEE, 2020, pp. 163–170.

[34] H. He, S. Zhao, Z. Tang, J. C. Ho, Y. Saad, and Y. Xi, *An Efficient Nonlinear Acceleration Method that Exploits Symmetry of the Hessian*, https://arxiv.org/abs/2210.12573, 2022.

[35] H. He, S. Zhao, Y. Xi, and J. Ho, *AGE: Enhancing the convergence on GANs using alternating extra-gradient with gradient extrapolation*, in NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications, 2021.

[36] H. He, S. Zhao, Y. Xi, J. Ho, and Y. Saad, *GDA-AM: On the effectiveness of solving minimax optimization via Anderson mixing*, in Proceedings of the International Conference on Learning Representations, 2022, https://openreview.net/forum?id=3YqeuCVwy1d.

[37] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, https://arxiv.org/abs/1512.03385, 2015.

[38] K. Jbilou and H. Sadok, *LU implementation of the modified minimal polynomial extrapolation method for solving linear and nonlinear systems*, IMA J. Numer. Anal., 19 (1999), pp. 549–561, https://doi.org/10.1093/imanum/19.4.549.

[39] K. C. Jea and D. M. Young, *Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods*, Linear Algebra Appl., 34 (1980), pp. 159–194.

[40] H. Kasai, *hiroyuki-kasai/GDLibrary*, https://github.com/hiroyuki-kasai/GDLibrary.

[41] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, Frontiers in Appl. Math. 16, SIAM, Philadelphia, 1995.

[42] C. T. Kelley, *Solving Nonlinear Equations with Newton's Method*, SIAM, Philadelphia, 2003, https://doi.org/10.1137/1.9780898718898.

[43] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, https://arxiv.org/abs/1412.6980, 2017.

[44] T. N. Kipf and M. Welling, *Semi-supervised Classification with Graph Convolutional Networks*, preprint, https://arxiv.org/abs/1609.02907, 2016.

[45] A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, 2009, https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[46] L. Lin and C. Yang, *Elliptic preconditioner for accelerating the self-consistent field iteration in Kohn–Sham density functional theory*, SIAM J. Sci. Comput., 35 (2013), pp. S277–S298, https://doi.org/10.1137/120880604.

[47] D. C. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Math. Program., 45 (1989), pp. 503–528.

[48] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization*, https://arxiv.org/abs/1711.05101, 2019.

[49] V. Mai and M. Johansson, *Anderson acceleration of proximal gradient methods*, in Proceedings of the 37th International Conference on Machine Learning, Proc. Mach. Learn. Res. 119, H. D. III and A. Singh, eds., PMLR, 2020, pp. 6620–6629, http://proceedings.mlr.press/v119/mai20a.html.

[50] V. V. Mai and M. Johansson, *Nonlinear acceleration of constrained optimization algorithms*, in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019, pp. 4903–4907, https://doi.org/10.1109/ICASSP.2019.8682962.

[51] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, *Automating the construction of internet portals with machine learning*, Inf. Retr., 3 (2000), pp. 127–163.

[52] G. Meurant and J. D. Tebbens, *Krylov Methods for Nonsymmetric Linear Systems: From Theory to Computations*, Springer Ser. Comput. Math. 57, Springer, New York, 2020.
[53] L. Meyer, C. Barrett, and P. Haasen, *New crystalline phase in solid argon and its solid solutions*, J. Chem. Phys., 40 (1964), pp. 2744–2745.
[54] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, Springer, New York, 2014.
[55] C. W. Oosterlee and T. Washio, *Krylov subspace acceleration of nonlinear multigrid with application to recirculating flows*, SIAM J. Sci. Comput., 21 (2000), pp. 1670–1690, https://doi.org/10.1137/S1064827598338093.
[56] M. L. Pasini, J. Yin, V. Reshniak, and M. K. Stoyanov, *Anderson acceleration for distributed training of deep learning models*, in SoutheastCon 2022, IEEE, 2022, pp. 289–295, https://doi.org/10.1109/SoutheastCon48659.2022.9763953.
[57] S. Pollock, L. G. Rebholz, and M. Xiao, *Anderson-accelerated convergence of Picard iterations for incompressible Navier–Stokes equations*, SIAM J. Numer. Anal., 57 (2019), pp. 615–637, https://doi.org/10.1137/18M1206151.
[58] M. Powell, *A new algorithm for unconstrained optimization*, in Nonlinear Programming, J. Rosen, O. Mangasarian, and K. Ritter, eds., Academic Press, New York, 1970, pp. 31–65, https://doi.org/10.1016/B978-0-12-597050-1.50006-3.
[59] P. Pulay, *Convergence acceleration of iterative sequences. The case of SCF iteration*, Chem. Phys. Lett., 73 (1980), pp. 393–398.
[60] P. Pulay, *Improved SCF convergence acceleration*, J. Comput. Chem., 3 (1982), pp. 556–560.
[61] H.-r. Fang and Y. Saad, *Two classes of multisecant methods for nonlinear acceleration*, Numer. Linear Algebra Appl., 16 (2009), pp. 197–221, https://doi.org/10.1002/nla.617.
[62] H. E. Robbins, *A stochastic approximation method*, Ann. Math. Statist., 22 (1951), pp. 400–407, https://api.semanticscholar.org/CorpusID:16945044.
[63] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelpha, 2003.
[64] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, SIAM, Philadelphia, 2011, https://doi.org/10.1137/1.9781611970739.
[65] R. B. Schnabel, *Quasi-Newton Methods Using Multiple Secant Equations*, Technical Report CU-CS-247-83, Department of Computer Science, University of Colorado at Boulder, 1983.
[66] D. Scieur, L. Liu, T. Pumir, and N. Boumal, *Generalization of quasi-Newton methods: Application to robust symmetric multisecant updates*, in Proceedings of the 24th International Conference on Artificial Intelligence and Statistics, Proc. Mach. Learn. Res. 130, A. Banerjee and K. Fukumizu, eds., PMLR, 2021, pp. 550–558, https://proceedings.mlr.press/v130/scieur21a.html.
[67] D. Scieur, E. Oyallon, A. d'Aspremont, and F. Bach, *Online Regularized Nonlinear Acceleration*, https://arxiv.org/abs/1805.09639, 2018.
[68] W. Shi, S. Song, H. Wu, Y. Hsu, C. Wu, and G. Huang, *Regularized Anderson acceleration for off-policy deep reinforcement learning*, in NeurIPS, 2019.
[69] D. A. Smith, W. F. Ford, and A. Sidi, *Extrapolation methods for vector sequences*, SIAM Rev., 29 (1987), pp. 199–233, https://doi.org/10.1137/1029042.
[70] S. Sreenivas, S Mandava, and C. Forster, *Pretraining BERT with Layer-wise Adaptive Learning Rates,* 2019, https://developer.nvidia.com/blog/pretraining-bert-with-layer-wise-adaptive-learning-rates.
[71] E. Stiefel, *Relaxationsmethoden bester strategie zur lösung linearer gleichungssysteme*, Comment. Math. Helv., 29 (1955), pp. 157–179, https://doi.org/10.1007/BF02564277.
[72] K. Sun, Y. Wang, Y. Liu, Y. Zhao, B. Pan, S. Jui, B. Jiang, and L. Kong, *Damped Anderson mixing for deep reinforcement learning: Acceleration, convergence, and stabilization*, in NeurIPS, 2021.
[73] A. Toth and C. T. Kelley, *Convergence analysis for Anderson acceleration*, SIAM J. Numer. Anal., 53 (2015), pp. 805–819.
[74] D. Vanderbilt and S. G. Louie, *Total energies of diamond* (111) *surface reconstructions by a linear combination of atomic orbitals method*, Phys. Rev. B, 30 (1984), pp. 6118–6130.
[75] P. K. W. Vinsome, *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, AIME, 1976, pp. 149–159.
[76] H. F. Walker and P. Ni, *Anderson acceleration for fixed-point iterations*, SIAM J. Numer. Anal., 49 (2011), pp. 1715–1735, https://doi.org/10.1137/10078356X.
[77] F. Wei, C. Bao, and Y. Liu, *Stochastic Anderson mixing for nonconvex stochastic optimization*, Adv. Neural Inf. Process. Syst., 34 (2021), pp. 22995–23008.

[78] F. Wei, C. Bao, and Y. Liu, *A class of short-term recurrence Anderson mixing methods and their applications*, in Proceedings of the International Conference on Learning Representations, 2022, https://openreview.net/forum?id=_X90SIKbHa.
[79] P. Wilmott, S. Howison, and J. Dewynne, *The Mathematics of Financial Derivatives: A Student Introduction*, Cambridge University Press, Cambridge, UK, 1995.
[80] Z. Yao, A. Gholami, S. Shen, K. Keutzer, and M. W. Mahoney, *Adahessian: An adaptive second order optimizer for machine learning*, in Proceedings of the AAAI Conference on Artificial Intelligence, 2021.
[81] J. Zhang, B. O'Donoghue, and S. Boyd, *Globally Convergent Type*-I *Anderson Acceleration for Non-smooth Fixed-point Iterations*, https://arxiv.org/abs/1808.03971, 2018.