Received: 14 October 2017 | Revised: 3 September 2018 | Accepted: 7 September 2018

DOI: 10.1002/nla.2216

RESEARCH ARTICLE

WILEY

Objective acceleration for unconstrained optimization

Asbjørn Nilsen Riseth

Mathematical Institute, University of Oxford, Oxford OX2 6GG, UK

Correspondence
Asbjørn Nilsen Riseth, Mathematical Institute, University of Oxford, Oxford OX2 6GG, UK.
Email: riseth@maths.ox.ac.uk

Funding information
EPSRC Centre for Doctoral Training in Industrially Focused Mathematical Modelling, Grant/Award Number: EP/L015803/1

Summary

Acceleration schemes can dramatically improve existing optimization procedures. In most of the work on these schemes, such as nonlinear generalized minimal residual (N-GMRES), acceleration is based on minimizing the ℓ_2 norm of some target on subspaces of $\mathbb{R}^n$. There are many numerical examples that show how accelerating general-purpose and domain-specific optimizers with N-GMRES results in large improvements. We propose a natural modification to N-GMRES, which significantly improves the performance in a testing environment originally used to advocate N-GMRES. Our proposed approach, which we refer to as O-ACCEL (objective acceleration), is novel in that it minimizes an approximation to the *objective function* on subspaces of $\mathbb{R}^n$. We prove that O-ACCEL reduces to the full orthogonalization method for linear systems when the objective is quadratic, which differentiates our proposed approach from existing acceleration methods. Comparisons with the limited-memory Broyden–Fletcher–Goldfarb–Shanno and nonlinear conjugate gradient methods indicate the competitiveness of O-ACCEL. As it can be combined with domain-specific optimizers, it may also be beneficial in areas where limited-memory Broyden–Fletcher–Goldfarb–Shanno and nonlinear conjugate gradient methods are not suitable.

KEYWORDS

acceleration, full orthogonalization method, nonlinear GMRES, optimization

1 | INTRODUCTION

Gradient-based optimization algorithms normally iterate based on tractable approximations to the objective function at a particular point. Acceleration algorithms aim to combine the strengths of existing solvers with information from previous iterates. We propose an acceleration scheme that can be used on top of existing optimization algorithms, which generates a subspace from previous iterates, over which it aims to optimize the objective function. We call the algorithm O-ACCEL, short for objective acceleration.

Our idea closely resembles the work of De Sterck,[1] which introduced the preconditioned nonlinear generalized minimal residual (N-GMRES) algorithm for optimization. By using a more appropriate target to accelerate the optimization than N-GMRES does, we show, with numerical examples, how O-ACCEL more efficiently accelerates the steepest descent algorithm. When optimizing an objective f, N-GMRES is used as an accelerator from the point of view of solving the nonlinear system $\nabla f(x) = 0$, which arises from the first-order condition of optimality. It uses the idea of Krylov subspace

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.
© 2018 The Authors. *Numerical Linear Algebra With Applications* published by John Wiley & Sons Ltd.

acceleration from Washio et al.[2,3] for solving nonlinear equations that arise from discretizations of partial differential equations. The name N-GMRES arises from the fact that steepest descent preconditioned N-GMRES is equivalent to the standard generalized minimal residual (GMRES) procedure for linear systems of equations (Washio et al.,[2] De Sterck[1]). A similar idea, also arising from nonlinear equations, was described by Anderson[4] in 1965. See the work of Walker et al.[5] for a note on the similarities of the methods and the work of Fang et al.,[6] which puts Anderson acceleration in the context of a Broyden-type approximation of the inverse Jacobian. Brune et al.[7] show, with many numerical examples, that N-GMRES and Anderson acceleration can greatly improve convergence on nonlinear systems when combined with an appropriate preconditioner (nonlinear solver). In the setting of optimization, De Sterck et al.[8,9] show large improvements in convergence by applying N-GMRES acceleration to the computation of tensor decompositions.

More recently, Scieur et al.[10] have developed another acceleration method for convex optimization denoted regularized nonlinear acceleration (RNA), which Cartis et al.[11] have extended to the nonconvex case. Acceleration techniques differ from one another in several ways, but for convex quadratic objectives, the Anderson, N-GMRES, and Scieur et al. algorithms all coincide (Cartis et al.[11]). These methods all minimize the ℓ_2 norm of some objective in $\mathbb{R}^n$, the space of the decision variable. The proposed algorithm in this manuscript instead aims to minimize the objective function over a subspace of $\mathbb{R}^n$. We believe this is a natural target to accelerate against, especially when the optimization procedure is seeking descent directions. For convex quadratic functions, we prove that O-ACCEL with a steepest descent preconditioner reduces to the full orthogonalization method (FOM[12]), a Krylov subspace procedure for solving linear systems. This differentiates our method from the other acceleration techniques, which are related to the GMRES algorithm for linear systems.

Due to the close similarity with the proposed algorithm and N-GMRES, this manuscript focuses on numerical comparisons to N-GMRES under the same testing conditions as used by De Sterck.[1] On the test set from the work of De Sterck,[1] our acceleration scheme compares favorably to N-GMRES, as well as implementations of the nonlinear conjugate gradient (N-CG) and limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) methods (Nocedal et al.[13]). Further tests on the CUTEst test problem set (Gould et al.[14]) show that L-BFGS is more applicable to these problems; however, O-ACCEL again performs better than N-GMRES.

The manuscript is organized as follows. Motivation for the algorithm, and discussion around it, is covered in Section 2. Numerical tests that show the efficiency of our proposed acceleration procedure applied to steepest descent are presented in Section 3. We conclude and discuss further potential work in Section 4.

2 | OPTIMIZATION ACCELERATION WITH O-ACCEL

To fix notation, consider a twice continuously differentiable function $f \in C^2(\mathbb{R}^n)$ that is bounded below and has at least one minimizer. We aim to find local minima of the optimization problem

$$\min_{x\in\mathbb{R}^n} f(x). \tag{1}$$

Let $\mathcal{M}(f, x)$ denote an optimization procedure for f with initial guess $x \in \mathbb{R}^n$. This optimization procedure can, for example, be the application of one steepest descent, or Newton, step. We will refer to $\mathcal{M}$ as the preconditioner because it is applied in the same fashion as a right preconditioner for iterative procedures of linear systems (Brune et al.[7]). Given a sequence of previously explored iterates $x^{(1)}, \ldots, x^{(k)}$ and a proposed new guess $x^P = \mathcal{M}(f, x^{(k)})$, we will try to accelerate the next iterate $x^{(k+1)}$ toward a minimizer. Define

$$\mathcal{K}_k^O\left(x^P\right) = \text{span}\left\{x^{(1)} - x^P, \ldots, x^{(k)} - x^P\right\}. \tag{2}$$

The acceleration step aims to minimize f over the subset $x^P + \mathcal{K}_k^O(x^P)$, which can be interpreted as a generalization from a line search to a hyperplane search. Let $\alpha \in \mathbb{R}^k$, and set

$$x^A(\alpha) = x^P + \sum_{j=1}^{k} \alpha_j \left(x^{(j)} - x^P\right). \tag{3}$$

Note that, when $k = 1$, minimizing f over $\mathcal{K}_k^O(x^P)$ is equivalent to the standard line search problem of minimizing $\lambda \mapsto f(x^{(1)} + \lambda(x^P - x^{(1)}))$. The first-order condition for α to be a minimizer of the function $\alpha \mapsto f(x^A(\alpha))$ is

$$\nabla_\alpha f\left(x^P + \sum\nolimits_{j=1}^{k} \alpha_j \left(x^{(j)} - x^P\right)\right) = 0. \tag{4}$$

Define the gradient $g(x) = \nabla_x f(x)$. For $l = 1, \ldots, k$,

$$\frac{\partial}{\partial \alpha_l} f\left(x^P + \sum_{j=1}^{k} \alpha_j \left(x^{(j)} - x^P\right)\right) = g\left(x^P + \sum_{j=1}^{k} \alpha_j \left(x^{(j)} - x^P\right)\right)^{\intercal} \left(x^{(l)} - x^P\right), \tag{5}$$

where superscript $\intercal$ denotes the transpose. The O-ACCEL algorithm aims to linearize the first-order condition $\nabla_\alpha f(x^A(\alpha)) = 0$ in the following way. Let $H(x)$ denote the Hessian of f at x. By linearizing $\alpha \mapsto g(x^A(\alpha))$, we get

$$\begin{aligned} g\left(x^P + \sum_{j=1}^{k} \alpha_j \left(x^{(j)} - x^P\right)\right) &\approx g\left(x^P\right) + H\left(x^P\right) \sum_{j=1}^{k} \alpha_j \left(x^{(j)} - x^P\right) \\ &= g\left(x^P\right) + H\left(x^P\right)\left(\boldsymbol{X} - \boldsymbol{X}^P\right)\alpha, \end{aligned} \tag{6}$$

where we use the matrices $\boldsymbol{X} = [x^{(1)}, \ldots, x^{(k)}] \in \mathbb{R}^{n \times k}$ and $\boldsymbol{X}^P = [x^P, \ldots, x^P] \in \mathbb{R}^{n \times k}$. Given this linearization, we aim to find an $\alpha \in \mathbb{R}^k$ that approximately satisfies the first-order condition. We can do this by combining (5) and (6), and then, look for an $\alpha \in \mathbb{R}^k$ that solves

$$\alpha^{\intercal}\left(\boldsymbol{X} - \boldsymbol{X}^P\right)^{\intercal} H\left(x^P\right)\left(x^{(l)} - x^P\right) = -g\left(x^P\right)^{\intercal}\left(x^{(l)} - x^P\right), \qquad l = 1, \ldots, k. \tag{7}$$

In matrix form, the system of equations becomes

$$\left(\boldsymbol{X} - \boldsymbol{X}^P\right)^{\intercal} H\left(x^P\right)\left(\boldsymbol{X} - \boldsymbol{X}^P\right)\alpha = -\left(\boldsymbol{X} - \boldsymbol{X}^P\right)^{\intercal} g\left(x^P\right). \tag{8}$$

There are cases where we may not wish to compute the Hessian of f explicitly, for example, if $\mathcal{M}$ does not use it. We can instead use an approximation $\tilde{H}(x^P)$ of the Hessian $H(x^P)$ or its action on vectors in $\mathcal{K}_k^O(x^P)$. The iterative Hessian approximation algorithms that are used in quasi-Newton methods can provide one avenue of research. In the numerical experiments provided in this manuscript, we instead focus on approximating the action of the Hessian on $\mathcal{K}_k^O(x^P)$ to first order by

$$H\left(x^P\right)\left(x^{(l)} - x^P\right) \approx g\left(x^{(l)}\right) - g\left(x^P\right). \tag{9}$$

Let $g(\boldsymbol{X}) = [g(x^{(1)}), \ldots, g(x^{(k)})]$, and define $g(\boldsymbol{X}^P)$ similarly. This gives a second approximation to the first-order conditions

$$\left(\boldsymbol{X} - \boldsymbol{X}^P\right)^{\intercal}\left(g(\boldsymbol{X}) - g\left(\boldsymbol{X}^P\right)\right)\alpha = -\left(\boldsymbol{X} - \boldsymbol{X}^P\right)^{\intercal} g\left(x^P\right). \tag{10}$$

In this manuscript, we investigate the performance of the objective-based acceleration using (10).

To contrast our work with the N-GMRES optimization algorithm in the work of De Sterck,[1] minimizing the ℓ_2 norm of the approximation of $g(x^A)$ established from (6) and (9) results in the linear least squares problem

$$\min_{\alpha \in \mathbb{R}^k} \left\| g\left(x^P\right) + \sum_{j=l}^{k} \alpha_l \left(g\left(x^{(l)}\right) - g\left(x^P\right)\right) \right\|_2. \tag{11}$$

Its solution can be found from the normal equation

$$\left(g(\boldsymbol{X}) - g\left(\boldsymbol{X}^P\right)\right)^{\intercal}\left(g(\boldsymbol{X}) - g\left(\boldsymbol{X}^P\right)\right)\alpha = -\left(g(\boldsymbol{X}) - g\left(\boldsymbol{X}^P\right)\right)^{\intercal} g\left(x^P\right). \tag{12}$$

We argue that the O-ACCEL algorithm is more appropriate for an optimization problem than N-GMRES. When we are restricted to subsets of the decision space, reduction in the value of the objective is a better indicator of moving toward a minimizer than reduction in the gradient norm. In effect, N-GMRES ignores the extra information provided by f. This is better illustrated in the case when $k = 1$, where it is standard to perform a line search on the objective rather than the gradient norm.

2.1 | Algorithm

The proposed acceleration procedure, which we call O-ACCEL, is described in Algorithm 1. The number of stored previous iterates w denotes the history size. Setting an upper bound on the history size can be necessary due to storage constraints or to prevent the local approximations of (6) and (9) from using iterates far away from x^P. If the direction from x^P to the accelerated step x^A is not a descent direction, it indicates that the linearized approximation around x^P is bad for the currently stored iterates. For simplicity, we therefore choose to reset the history size to $w = 1$ when we encounter such cases.

Algorithm 1 The O-ACCEL algorithm

1: **procedure** OACCEL($x^{(1)}, \dots, x^{(w)}$)
2: $x^P \leftarrow \mathcal{M}(f, x^{(w)})$
3: Approximate $H(x^P) \approx \tilde{H}$, or its action
4: $A \leftarrow (\boldsymbol{X} - \boldsymbol{X}^P)^\intercal \tilde{H} (\boldsymbol{X} - \boldsymbol{X}^P)$ Section 3: $A \leftarrow (\boldsymbol{X} - \boldsymbol{X}^P)^\intercal (g(\boldsymbol{X}) - g(\boldsymbol{X}^P))$
5: $b \leftarrow -(\boldsymbol{X} - \boldsymbol{X}^P)^\intercal g(x^P)$
6: Solve $A\alpha = b$
7: $x^A \leftarrow x^P + \sum_{j=1}^{w} \alpha_j (x^{(j)} - x^P)$
8: **if** $x^A - x^P$ is a descent direction **then**
9: $x^{(w+1)} \leftarrow$ linesearch$(x^P + \lambda(x^A - x^P))$
10: `reset` $\leftarrow$ `false`
11: **else**
12: $x^{(w+1)} \leftarrow x^P$
13: `reset` $\leftarrow$ `true`
14: **end if**
15: **return** $(x^{(w+1)}$, `reset`$)$
16: **end procedure**

To prevent recomputation of $g(x^{(j)})$ for $j = 1, \dots, w$ in each application of the procedure, we store these vectors for later use. The computational cost of the algorithm is approximately the same as w-history L-BFGS with two-loop recursion (De Sterck[1]). In terms of storage, O-ACCEL and L-BFGS both store $2w$ vectors of size n. In addition, our implementation of O-ACCEL, as described in Algorithm 2 below, reduces the number of flops required by storing a $w \times w$ matrix of previously calculated values. For the numerical experiments, we have used $w = 20$, in accordance with the work of De Sterck.[1] It was, however, shown in the work of De Sterck[1] that N-GMRES can already provide good results with $w = 3$. Tests using O-ACCEL with $w = 5$, although not included here, provide almost as good results as reported in Section 3. Note that, if the Hessian is sparse, it may be more storage efficient to find α from the linear system in (8) than using a large w.

Remark 1. The RNA approach suggested by Scieur et al.[10] is called separately from the iterations by the optimizer, when judged appropriate. This contrasts with Algorithm 1, where the acceleration happens at each iteration. The O-ACCEL acceleration can be applied separately in the same fashion as in the work of Scieur et al.[10]; however, this is not considered in this manuscript.

2.2 | O-ACCEL as a FOM

The optimality condition (5) for the function $\alpha \mapsto f(x^A(\alpha))$ is $g(x^A)^\intercal (x^{(l)} - x^P) = 0$, for $l = 1, \dots, k$. Hence, we look for $x^A \in x^P + \mathcal{K}_k^O(x^P)$ so that $g(x^A) \perp \mathcal{K}_k^O(x^P)$. This condition reduces to FOM (Saad[12]) when $g(x)$ is linear and $\mathcal{M}(f, x)$ is a steepest descent algorithm. When the Hessian is symmetric positive definite, FOM is mathematically equivalent to the conjugate gradient method. We can therefore think of O-ACCEL as a N-CG method that approximates the orthogonality condition with a larger history size.

The FOM is an iterative procedure for solving a linear system $Ax = b$. With initial guess $x^{(1)}$ and residual $r^{(1)} = b - Ax^{(1)}$, define the Krylov subspace

$$\mathcal{K}_k\left(A, r^{(1)}\right) = \text{span}\left\{r^{(1)}, Ar^{(1)}, \dots, A^{k-1}r^{(1)}\right\}. \tag{13}$$

The iterate $x^{(k+1)}$ of FOM is an element in $x^{(1)} + \mathcal{K}_k(A, r^{(1)})$ such that $b - Ax^{(k+1)} \perp \mathcal{K}_k(A, r^{(1)})$.

For convex quadratic objectives $f(x) = \frac{1}{2}x^\intercal Ax - x^\intercal b$, the gradient $g(x) = Ax - b$ is linear and the optimum must satisfy the equation $Ax = b$. The residuals $r^{(k)} = b - Ax^{(k)}$ are equal to the negative gradient $-g(x^{(k)})$. Therefore, O-ACCEL with a steepest descent preconditioner yields $x^P = \mathcal{M}(f, x^{(k)}) = x^{(k)} + \lambda^{(k)} r^{(k)}$ for some $\lambda^{(k)} > 0$.

Theorem 1. *Let $\mathcal{M}$ be a steepest descent preconditioner and $f(x) = \frac{1}{2}x^\intercal Ax - x^\intercal b$. Let the O-ACCEL algorithm take the step $x^{(w+1)} = x^A$ in Line 9 of Algorithm 1. Then, the iterates of the O-ACCEL algorithm form the FOM sequence of the linear system $Ax = b$.*

Algorithm 2 Implementation of O-ACCEL algorithm. Indentation and curly brackets denote scope

Input: $f, g, \mathcal{M}, x, w_{\max}, \epsilon_0$, tolerance description
Output: x satisfying tolerance description

while Not reached tolerance **do**
  $\mathtt{x}_1 \leftarrow x$; $\mathtt{r}_1 \leftarrow g(x)$; $\mathtt{q}_{11} \leftarrow x^\intercal \mathtt{r}_1$
  $w \leftarrow 1$; $k \leftarrow 0$; $\mathtt{reset} \leftarrow \mathtt{false}$
  while `reset` is `false` **do**
    $k \leftarrow k + 1$
    $x \leftarrow \mathcal{M}(f, x)$; $r \leftarrow g(x)$
    if reached tolerance **then**
      break
    end if
    $\eta \leftarrow x^\intercal r$
    for $i = 1, \ldots, w$ { $\xi_i^{(1)} \leftarrow \mathtt{x}_i^\intercal r$; $\xi_i^{(2)} \leftarrow x^\intercal \mathtt{r}_i$; $\mathtt{b}_i \leftarrow \eta - \xi_i^{(1)}$ }
    for $i = 1, \ldots, w$ { **for** $j = 1, \ldots, w$ { $\mathtt{A}_{ij} \leftarrow \mathtt{q}_{ij} - \xi_i^{(1)} - \xi_j^{(2)} + \eta$ } }
    $\epsilon \leftarrow \epsilon_0 \cdot \max\{\mathtt{A}_{11}, \ldots, \mathtt{A}_{ww}\}$
    Solve $\begin{pmatrix} \mathtt{A}_{11} + \epsilon & \cdots & \mathtt{A}_{1w} \\ \vdots & \ddots & \vdots \\ \mathtt{A}_{w1} & \cdots & \mathtt{A}_{ww} + \epsilon \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_w \end{pmatrix} = \begin{pmatrix} \mathtt{b}_1 \\ \vdots \\ \mathtt{b}_w \end{pmatrix}$
    $x^A \leftarrow x + \sum_{i=1}^{w} \alpha_i (\mathtt{x}_i - x)$
    $d \leftarrow x^A - x$
    if $d^\intercal r \geq 0$ **then**
      $\mathtt{reset} \leftarrow \mathtt{true}$
    else
      $x \leftarrow \text{linesearch}(x + \lambda d)$
      $w \leftarrow \min(w + 1, w_{\max})$
      $j \leftarrow (k \mod w_{\max}) + 1$
      $\mathtt{x}_j \leftarrow x$
      $\mathtt{r}_j \leftarrow g(x)$
      for $i = 1, \ldots, w$ { $\mathtt{q}_{ij} \leftarrow \mathtt{x}_i^\intercal \mathtt{r}_j$; $\mathtt{q}_{ji} \leftarrow \mathtt{x}_j^\intercal \mathtt{r}_i$ }
    end if
  end while
end while

We shall shortly prove the theorem after deriving new expressions for $\mathcal{K}_k(A, r^{(1)})$. First, note that, for any x, a reordering of terms can show that

$$\mathcal{K}_k^O(x) = \text{span}\left\{x - x^{(k)}, x^{(k)} - x^{(k-1)}, \ldots, x^{(2)} - x^{(1)}\right\}, \tag{14}$$

$$x + \mathcal{K}_k^O(x) = x^{(1)} + \text{span}\left\{x - x^{(k)}, x^{(k)} - x^{(k-1)}, \ldots, x^{(2)} - x^{(1)}\right\}. \tag{15}$$

This motivates the next lemma, which connects the space on the right-hand side of (14) to $\mathcal{K}_{k+1}(A, r^{(1)})$.

Lemma 1. *Let $x^{(1)}, \ldots, x^{(k)}$ be a given sequence of FOM iterates for a linear system $Ax = b$. Assume that* span$x^{(k)} - x^{(k-1)}, \ldots, x^{(2)} - x^{(1)} = \mathcal{K}_k(A, r^{(1)})$*, and let $x^P = x^{(k)} + \lambda r^{(k)}$ for some $\lambda > 0$. Then,*

$$\text{span}\left\{x^P - x^{(k)}, x^{(k)} - x^{(k-1)}, \ldots, x^{(2)} - x^{(1)}\right\} = \mathcal{K}_{k+1}\left(A, r^{(1)}\right). \tag{16}$$

Proof. By definition of x^P and the properties of the FOM sequence,

$$x^P - x^{(k)} = \lambda r^{(k)} \perp \mathcal{K}_k\left(A, r^{(1)}\right). \tag{17}$$

As $x^{(k)} \in x^{(1)} + \mathcal{K}_k(A, r^{(1)})$, we have $r^{(k)} \in \mathcal{K}_{k+1}(A, r^{(1)})$ because

$$r^{(k)} \in b - A\left(x^{(1)} + \mathcal{K}_k\left(A, r^{(1)}\right)\right) = r^{(1)} - A\mathcal{K}_k\left(A, r^{(1)}\right) \in \mathcal{K}_{k+1}\left(A, r^{(1)}\right). \tag{18}$$

Therefore, $\text{span}\{r^{(k)}, \mathcal{K}_k(A, r^{(1)})\} = \mathcal{K}_{k+1}(A, r^{(1)})$. This equality yields the result by replacing $r^{(k)}$ and $\mathcal{K}_k(A, r^{(1)})$ with (17) and $\text{span}\{x^{(k)} - x^{(k-1)}, \dots, x^{(2)} - x^{(1)}\}$. □

Proof of Theorem 1. We prove the result by induction on the sequence $x^{(1)}, \dots, x^{(k)}$ arising from the O-ACCEL algorithm. Let $k = 2$; then,

$$x^{(2)} = x^P + \alpha^{(1)}\left(x^{(1)} - x^P\right) \tag{19}$$

$$= x^{(1)} + \lambda^{(1)} r^{(1)} - \alpha^{(1)}\lambda^{(1)} r^{(1)} \in x^{(1)} + \mathcal{K}_1\left(A, r^{(1)}\right). \tag{20}$$

Therefore, $\text{span}\{x^{(2)} - x^{(1)}\} = \mathcal{K}_1(A, r^{(1)})$. From (5), the residual $b - Ax^{(2)} \perp x^P - x^{(1)} = \lambda^{(k)} r^{(1)}$, and thus, $x^{(2)}$ is the second FOM iterate. This establishes the base case for the induction proof.

The inductive step follows from Lemma 1 together with (14) and (15) and, hence, proves that the O-ACCEL iterates are the FOM iterates for $Ax = b$. □

Remark 2. The connection to the FOM differentiates O-ACCEL from N-GMRES, Anderson acceleration, and RNA, which reduce to GMRES for quadratic objectives.

3 | NUMERICAL EXPERIMENTS

In order to investigate the performance of the proposed algorithm, we implement it with two preconditioners $\mathcal{M}$. The first is steepest descent with line search, and the second is steepest descent with a fixed step length. They are compared with the N-GMRES algorithm with the same preconditioners, and implementations of the N-CG variant with the Polak–Ribière update formula, and the two-loop recursion version of the L-BFGS method (Nocedal et al.[13]). The test problems considered in Sections 3.1 to 3.4 are the same eight problems that were used in the work of De Sterck[1] to advocate N-GMRES. We also include experiments from 33 CUTEst problems to further test the applicability of the algorithms. The results are presented in the form of performance profiles, as introduced by Dolan et al.[15] based on the number of function/gradient evaluations.

The main focus of this manuscript is to compare the performance of the proposed algorithm to the N-GMRES algorithm. To this end, we have used the MATLAB implementation of this algorithm, available online.* The O-ACCEL implementation, and the rest of the code required to generate the test result data, is also made available by the author.† Our implementation of O-ACCEL follows the exact same steps, only replacing the calculations needed to solve the N-GMRES system in (12) with those of the linear system in (10). The implementation is detailed in Algorithm 2. It closely follows the instructions from the work of Washio et al.,[2] including a regularization for the linear system.

The regularization is used to prevent the direct linear solver we use to find α from crashing when A is ill conditioned or singular, which can happen if the vectors $g(x^{(k)}) - g(x^P)$ are linearly dependent. Let $A \in \mathbb{R}^{w\times w}$ denote the system matrix $(\boldsymbol{X} - \boldsymbol{X}^P)^\intercal(g(\boldsymbol{X}) - g(\boldsymbol{X}^P))$. Then, for some tolerance $\epsilon_0 > 0$, set $\epsilon = \epsilon_0 \cdot \max\{A_{ii}\}_{i=1}^w$. The max term is used to scale the regularization in accordance with the optimization problem. With $I \in \mathbb{R}^{w\times w}$ as the identity matrix, we solve the linear problem

$$(A + \epsilon I)\alpha = b, \tag{21}$$

rather than the linear problem $A\alpha = b$, as defined in Algorithm 1. This is a Tikhonov-type regularization,[16] often employed to regularize ill-conditioned problems. Washio et al.[2] show that the error in the resulting α is negligible for the N-GMRES problem (12), provided ϵ is much smaller than the smallest nonzero eigenvalue of the system matrix. The error for the O-ACCEL system can be analyzed within a general Tikhonov regularization framework; see, for example, the work of Neumaier.[16] We do not investigate the impact of the regularization parameter further in this manuscript and use the value $\epsilon_0 = 10^{-14}$ that was used in the N-GMRES code by De Sterck.[1]

For the remainder of the section, we present the test problems, provide details for the parameter choices, and discuss the test results.

*http://www.hansdesterck.net/Publications-by-topic/nonlinear-preconditioning-for-nonlinear-optimization
†https://github.com/anriseth/objective_accel_code

3.1 | Test problems from De Sterck

We describe the seven test problems from the work of De Sterck.[1] All the functions are defined as $f : \mathbb{R}^n \to \mathbb{R}$, and the matrices mentioned are all in $\mathbb{R}^{n\times n}$.

Problem A. Quadratic objective function with symmetric positive-definite diagonal matrix D

$$\begin{aligned} f(x) &= \frac{1}{2}(x-x^*)^\intercal D(x-x^*), \text{ where} \\ D &= \text{diag}(1, 2, \ldots, n) \text{ and} \\ x^* &= [1, \ldots, 1]. \end{aligned} \tag{22}$$

The minimizer x^* of Problem A is unique, with $f(x^*) = 0$. The gradient is given by $g(x) = D(x - x^*)$.

Problem B. Problem A with paraboloid coordinate transformation

$$\begin{aligned} f(x) &= \frac{1}{2}y(x-x^*)^\intercal Dy(x-x^*), \text{ where} \\ D &= \text{diag}(1, 2, \ldots, n), \\ x^* &= [1, \ldots, 1], \text{ and} \\ y_1(z) &= z_1 \text{ and } y_j(z) = z_j - 10\, z_1^2 \quad (i = 2, \ldots, n). \end{aligned} \tag{23}$$

The minimizer is again x^*, with $f(x^*) = 0$. The gradient is $g(x) = Dy(x - x^*) - 20(x_1 - x_1^*) \times (\sum_{j=2}^n (Dy(x-x^*))_j)$ $[1, 0, \ldots, 0]^\intercal$.

Problem C. Problem B with a random nondiagonal matrix T with condition number n

$$\begin{aligned} f(x) &= \frac{1}{2}y(x-x^*)^\intercal Ty(x-x^*), \text{ where} \\ x^* &= [1, \ldots, 1], \\ y_1(z) &= z_1 \text{ and } y_j(z) = z_j - 10z_1^2 \quad (i = 2, \ldots, n), \text{ and} \\ T &= Q\,\text{diag}(1, 2, \ldots, n)\, Q^\intercal, \end{aligned} \tag{24}$$

where Q is a random orthogonal matrix. As in Problems A and B, the minimizer is x^* with $f(x^*) = 0$. The gradient is $g(x) = Ty(x - x^*) - 20(x_1 - x_1^*) \times (\sum_{j=2}^n (Ty(x-x^*))_j)[1, 0, \ldots, 0]^\intercal$.

Problem D. Extended Rosenbrock function (problem (21) from the work of Moré et al.[17])

$$\begin{aligned} f(x) &= \frac{1}{2}\sum_{j=1}^n t_j(x)^2, \text{ where } n \text{ is even,} \\ t_j &= 10\left(x_{j+1} - x_j^2\right) \qquad (j \text{ odd}), \text{ and} \\ t_j &= 1 - x_{j-1} \qquad\qquad (j \text{ even}). \end{aligned} \tag{25}$$

The unique minimum $f(x^*) = 0$ is attained at $x^* = [1, \ldots, 1]$. The derivative can be computed using $g_k(x) = \sum_{j=1}^n t_j \frac{\partial t_j}{\partial x_k}$, $(k = 1, \ldots, n)$. Gradients for Problems E–G can be computed in similar fashion.

Problem E. Extended Powell singular function (problem (22) the work of from Moré et al.[17])

$$\begin{aligned} f(x) &= \frac{1}{2}\sum_{j=1}^n t_j(x)^2, \text{ where } n \text{ is a multiple of 4,} \\ t_{4j-3} &= x_{4j-3} + 10x_{4j-2}, \\ t_{4j-2} &= \sqrt{5}(x_{4j-1} - x_{4j}), \\ t_{4j-1} &= (x_{4j-2} - 2x_{4j-1})^2, \\ t_{4j} &= \sqrt{10}(x_{4j-3} - x_{4j})^2 \qquad \text{for } j = 1, \ldots, n/4. \end{aligned} \tag{26}$$

The unique minimum $f(x^*) = 0$ is attained at $x^* = 0$.

Problem F. The trigonometric function (problem (26) from the work of Moré et al.[17])

$$f(x) = \frac{1}{2}\sum_{j=1}^{n} t_j(x)^2, \text{ where}$$
$$t_j = n + j(1 - \cos x_j) - \sin x_j - \sum_{i=1}^{n} \cos(x_i). \tag{27}$$

The unique minimum $f(x^*) = 0$ is attained at $x^* = 0$. Note that, in the work of De Sterck,[1] a minus sign is used in front of $j(1 - \cos x_j)$. We follow the original formulation of Moré et al.[17]

Problem G. Penalty function I (problem (23) from the work of Moré et al.[17])

$$f(x) = \frac{1}{2}\left(t_0(x)^2 + \sum_{j=1}^{n} t_j(x)^2\right), \text{ where}$$
$$t_0 = -0.25 + \sum_{j=1}^{n} x_j^2 \text{ and} \tag{28}$$
$$t_j = \sqrt{10^{-5}}(x_j - 1) \qquad (j = 1, \ldots, n).$$

The minimum is not known explicitly for Problem G and depends on the value of n.

3.2 | Experiment design

We test the N-GMRES and O-ACCEL algorithms with two steepest descent preconditioners

$$\mathcal{M}_{\mathrm{Z}}(f, x) = x - \lambda_{\mathrm{Z}} \frac{g(x)}{\|g(x)\|_2}, \qquad \text{with Z} = \mathrm{A},\ \mathrm{B},\ \text{and}$$
$$\lambda_{\mathrm{A}} = \text{determined by line search}, \tag{29}$$
$$\lambda_{\mathrm{B}} = \min(\delta, \|g(x)\|_2). \tag{30}$$

Thus, the two preconditioners only differ in the choice of step length. Option A employs a globalizing strategy with a chosen line search, whereas option B takes a predetermined step length. By choosing a short predetermined step length $\delta > 0$, we expand the subspace to search for α and stay close to the previous iterate $x^{(k)}$, hopefully improving the linearizations in (6) and (9). For the experiments, we use the line search algorithm by Moré et al.,[18] which satisfies the Wolfe conditions (Nocedal et al.[13]). It is both employed for $\mathcal{M}_{\mathrm{A}}$ and in the line search $x^P + \lambda(x^A - x^P)$ between the preconditioned step x^P and the accelerated step x^A of the N-GMRES and O-ACCEL routines.

To closely follow the testing conditions of De Sterck,[1] we use the N-CG, L-BFGS, and Moré–Thuente line search implementations from the Poblano toolbox by Dunlavy et al.[19] These may not be state-of-the-art implementations; however, the main focus of this manuscript is to investigate the performance of the N-GMRES and O-ACCEL algorithms. Future work will include testing the O-ACCEL algorithm with appropriate preconditioners on more comprehensive test sets, against state-of-the-art implementations of gradient-based optimization algorithms.

All optimization procedures employ the Moré–Thuente line search with the following options: decrease tolerance $c_1 = 10^{-4}$ and curvature tolerance $c_2 = 0.1$ for the Wolfe conditions, starting step length $\lambda = 1$ and a maximum of 20 f/g evaluations. The N-GMRES and O-ACCEL history lengths are set to $w_{\max} = 20$, and the regularization parameter is set to $\epsilon_0 = 10^{-12}$. For $\mathcal{M}_{\mathrm{B}}$, the fixed step length is set to $\delta = 10^{-4}$. The L-BFGS history size is set to 5. Larger history sizes were found by De Sterck[1] to be harmful for the L-BFGS performance on this test set.

Note that our choice of curvature tolerance $c_2 = 0.1$ is different from that of De Sterck,[1] where $c_2 = 0.01$ was used. There are two reasons for this. First, our choice is often used in practice (see ch. 3.1 of the work of Nocedal et al.[13]), and it reduces the number of function evaluations for all the solvers considered. Second, we are interested in comparing the outer solvers; however, smaller values of c_2 moves work from the outer solvers to the line search algorithms.

We test Problem A–C for both problem sizes $n = 100$ and $n = 200$. Problem D is tested with $n = 500$, 1,000, 50,000, 100,000. Problem E with $n = 100$, 200, 50,000, 100,000. Problem F is called with $n = 200, 500$, and finally, Problem G with $n = 100, 200$. Each combination of problem and problem size is run 1,000 times, with the components of the initial guess drawn uniformly random from the interval $[0, 1]$. For Problem C, each instance of the problem generates a

new random orthogonal matrix Q. This results in 18,000 individual tests for the comparison. To evaluate performance, we count the number of objective evaluations required for the algorithms to reach an iterate x such that $f(x) - f^* < 10^{-10}(f(x^{(0)}) - f^*)$. A solver run is labeled as failed if it does not reach tolerance within 1,500 iterations. The minimum value f^* is known for Problems A–F; however, for Problem G, we estimate f^* using the lowest value attained across all the optimization procedures. The results on the collection of 18,000 test instances are discussed in Section 3.3, whereas the Appendix provides tables of results on the individual problems and problem sizes.

Note that our reporting of the numerical experiments differs from that of De Sterck[1] in two ways: First, we run each problem combination 1,000 times, instead of 10 times. Second, we evaluate the results based on performance profiles and tables of quantiles, instead of solely reporting the average number of evaluations to reach tolerance. We believe the high number of test runs is important for more consistent values of the statistics reported in the Appendix across computers, further stabilized by using quantiles rather than averages.

3.3 | Performance profiles

In order to evaluate the performance of optimizers on test sets with problems of varying size and difficulty, Dolan et al.[15] proposed the use of performance profiles. For completeness, we first define the performance profile for our chosen metric of objective evaluations. Let $\mathcal{P}$ denote the test set of the $n_p = 18{,}000$ problems, and let n_s denote the number of solvers. For each problem $p \in \mathcal{P}$ and solver s, define

$$t_{p,s} = \text{number of } f \text{ evaluations required to reach tolerance.} \tag{31}$$

In the numerical tests, we say that the solver has reached tolerance for the problem when the relative decrease in the objective value is at least 10^{-10}, that is,

$$t_{p,s} = \min\left\{k \geq 1 | f\left(x^{(k)}\right) - f^* < 10^{-10}\left(f\left(x^{(0)}\right) - f^*\right)\right\}. \tag{32}$$

Remark 3. Note that the numbers of objective and gradient calls are the same for each of the optimizers considered in this manuscript. This is due to the use of the Moré–Thuente line search algorithm.

Let $\underline{t}_p$ denote the lowest number of f evaluations needed to reach tolerance for problem p across all the solvers:

$$\underline{t}_p = \min\{t_{p,s} | 1 \leq s \leq n_s\}. \tag{33}$$

The performance ratio measures the performance on problem p by solver s, as defined by

$$\rho_{p,s} = t_{p,s}/\underline{t}_p. \tag{34}$$

The value is bounded below by 1, and $\rho_{p,s} = 1$ for at least one solver s. If solver s does not solve problem p, then we set $\rho_{p,s} = \infty$. We define the performance profile $p_s : [1, \infty) \to [0, 1]$, for solver s, by

$$p_s(\tau) = \frac{1}{n_p}\text{size}\{p \in \mathcal{P} | \rho_{p,s} \leq \tau\}. \tag{35}$$

The performance profile for a solver s can be viewed as an empirical cumulative "distribution" function representing the probability of the solver s reaching tolerance within a ratio τ of the fastest solver for each problem. In particular, $p_s(1)$ gives the proportion of problems for which solver s performed best. For large values of τ, the performance profile $p_s(\tau)$ indicates robustness, that is, what proportion of all the test problems were solved by the solver.

Figure 1 plots the performance profile of the $n_s = 6$ solvers considered: N-CG, L-BFGS, and N-GMRES and O-ACCEL with steepest descent preconditioning using both a line search (A) and a fixed step size (B). It is clear that O-ACCEL-B and L-BFGS are the best performers across the test set. For 44% of the test problems, they reach tolerance in the fewest f evaluations, and they also solve the largest proportion of problems within higher factors τ of the best performance ratio. There is also a region where N-CG does particularly well, solving the largest proportion of problems within two to three times the highest performing solver. The worst performers are N-GMRES-A and O-ACCEL-A, mainly due to the high amount of work that the line search must do to satisfy the Wolfe conditions along the steepest descent directions.

It is notable that O-ACCEL-B is competitive with L-BFGS on the test set. Tests, not presented in this work, indicate that the L-BFGS performance improves by using a line search with Wolfe curvature condition parameter $c_2 = 0.9$, rather than $c_2 = 0.1$, as used in this manuscript. The main focus of this manuscript is, however, to investigate the potential improvement of minimizing the objective rather than an ℓ_2 norm of the gradient. Thus, we are more interested in the comparison between N-GMRES and O-ACCEL. The two plots in Figure 2 show the performance profiles comparing N-GMRES and

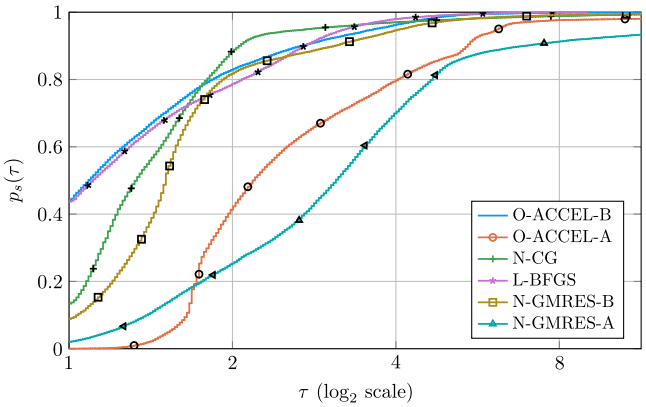


FIGURE 1 Performance profiles, defined in (35), for Problems A–G. O-ACCEL preconditioned with a fixed-step steepest descent (B) and L-BFGS mostly outperform the rest, except for higher factors of τ. They are also more robust, solving the largest proportion of the problems when the computational budget is large. O-ACCEL = objective acceleration; N-CG = nonlinear conjugate gradient; L-BFGS = limited-memory Broyden–Fletcher–Goldfarb–Shanno; N-GMRES = nonlinear generalized minimal residual

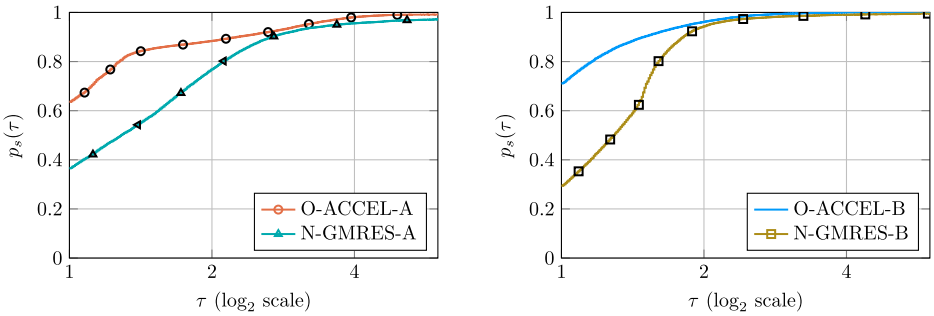
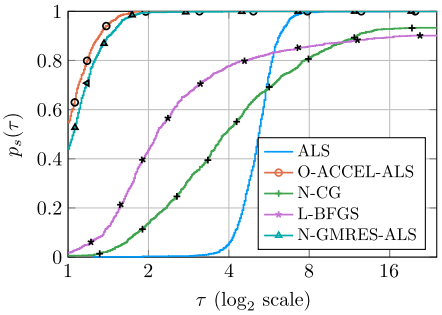


FIGURE 2 Performance profiles comparing N-GMRES and O-ACCEL with steepest descent (left) with line search (a) and (right) without (b). O-ACCEL outperforms N-GMRES in both cases on our test set. Note that the lines of O-ACCEL-A and N-GMRES-A cross in Figure 1, but not in the left figure here, because the performance profiles change depending on the set of solvers considered. O-ACCEL = objective acceleration; N-GMRES = nonlinear generalized minimal residual

O-ACCEL and, in both cases, show a significant improvement by minimizing the objective. In fact, O-ACCEL reaches tolerance first on 63 % to 71 % of the test problems. The instances where N-GMRES does better were primarily in Problems E, F, and G, as can be seen from Table A2 in the Appendix. One of the findings of De Sterck[1] was that N-GMRES with line search-steepest descent often stagnated or converged very slowly. From the left plot of Figure 2, we see that this issue is reduced with the O-ACCEL acceleration. It also turns out that O-ACCEL-A has a larger success rate over the test set than N-GMRES-A.

3.4 | The tensor optimization problem from De Sterck

N-GMRES was initially developed to improve the speed of convergence on a tensor optimization problem.[8] De Sterck[8] and De Sterck[1] show that using N-GMRES with a domain-specific alternating least squares (ALS) algorithm preconditioner is better than generic optimizers such as L-BFGS and N-CG. De Sterck[1] states that "In this problem, a rank-three canonical tensor approximation (with 450 variables) is sought for a three-way data tensor of size $50 \times 50 \times 50$. The data tensor is generated starting from a canonical tensor with specified rank and random factor matrices that are modified to have prespecified column colinearity, and noise is added. This is a standard canonical tensor decomposition test problem (Acar et al.[20])." For this manuscript, we run the 1,000 realizations of the test problem using the code provided by De Sterck[1] with the parameter values described in Section 3.2. The algorithms tested for this problem are vanilla ALS, N-GMRES-ALS, O-ACCEL-ALS, N-CG, and L-BFGS. Figure 3a and Figure 3b show the performance profiles and quantiles for the number of f evaluations required to reach tolerance. We see that O-ACCEL-ALS and N-GMRES-ALS perform better than the other algorithms, which underscores the advantage of applying these acceleration methods to domain-specific algorithms.



(a)

Algorithm	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$
ALS	878	1,107	1,212
O-ACCEL-ALS	170	227	297
N-CG	405	807	2,450
L-BFGS	302	438	3,037
N-GMRES-ALS	169	234	312

(b)

FIGURE 3 Numerical results from the tensor optimization test problem. O-ACCEL and N-GMRES perform significantly better than the other solvers. (a) Performance profiles. (b) f evaluations. ALS = alternating least squares algorithm; O-ACCEL = objective acceleration; N-CG = nonlinear conjugate gradient; L-BFGS = limited-memory Broyden–Fletcher–Goldfarb–Shanno; N-GMRES = nonlinear generalized minimal residual

3.5 | CUTEst test problems

The test problems we have considered so far were taken from the work of De Sterck[1] and originally used to promote N-GMRES. We finish by presenting results from a numerical experiment using problems from the CUTEst problem set.[14] For this experiment, we compare the solvers O-ACCEL-B, L-BFGS, and N-GMRES-B, with the parameter values described in Section 3.2. The minima are not known for many of the CUTEst problems, and so we change the tolerance criterion to be defined in terms of the relative decrease of the gradient norm. The performance measure used for this experiment is

$$t_{p,s} = \min\left\{k \geq 1 \mid \left\|g\left(x^{(k)}\right)\right\|_\infty \leq 10^{-8}\left\|g\left(x^{(0)}\right)\right\|_\infty\right\}. \tag{36}$$

A solver run is labeled as failed if it does not reach tolerance within 2,000 iterations.

We run the experiment using implementations of the solvers from the package Optim (Mogensen et al.[21]) of the Julia programming language (Bezanson et al.[22]). To be sure, we have also verified that the Optim code yields the same results as the MATLAB code for Problems A–G.

The 33 problems we consider are listed in Tables A3 and A4 of the Appendix together with the results of the numerical experiment. We selected the problems with dimension $n = 50$ to 10,000 that satisfy the two criteria: (i) the objective type is in the category "other", and (ii) at least one of the solvers succeeds in reaching tolerance. Figure 4 shows performance profiles from the experiment. L-BFGS reaches tolerance first for most of the problems; however, O-ACCEL reaches tolerance within 2,000 iterations for more of the test problems. In the problems where L-BFGS does not reach tolerance, it stops because it fails prematurely, whereas N-GMRES-B only fails due to reaching 2,000 iterations. We believe the poorer performance of the acceleration algorithms for the CUTEst problems, compared with the previous experiments, is due to the poor performance of the steepest descent preconditioner on these problems. Again, O-ACCEL-B performs better than N-GMRES-B, which underscores our claim that accelerating based on the objective function is better than accelerating based on the gradient norm.

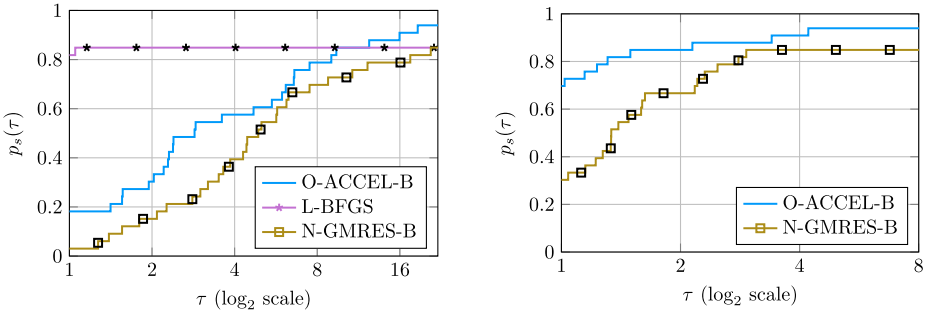


FIGURE 4 Performance profiles for the CUTEst test problems from Tables A3 and A4. L-BFGS is the highest performing one most of the time; however, O-ACCEL-B reaches tolerance for more problems. From the right-hand performance profile, we see that O-ACCEL-B outperforms N-GMRES-B on 70% of the problems. O-ACCEL = objective acceleration; L-BFGS = limited-memory Broyden–Fletcher–Goldfarb–Shanno; N-GMRES = nonlinear generalized minimal residual

4 | CONCLUSION

We have proposed a simple acceleration algorithm for optimization, based on the N-GMRES algorithm by Washio et al.[2] and De Sterck.[1] N-GMRES for optimization aims to accelerate a solver step when solving the nonlinear system $\nabla f(x) = 0$ by minimizing the residual in the ℓ_2 norm over a subspace from previous iterates. The acceleration step consists of solving a small linear system that arises from a linearization of the gradient.

We propose to take advantage of the structure of the optimization problem and instead accelerate based on the objective value $f(x)$. This new approach, labeled O-ACCEL, shows a significant improvement to the original N-GMRES algorithm in numerical tests when accelerating a steepest descent solver. The first test problems are taken from the work of De Sterck[1] and run under the same conditions that proved to be beneficial for N-GMRES. Further tests on a selection of CUTEst problems strengthen the conclusion that O-ACCEL outperforms N-GMRES. Another strength of these acceleration algorithms is that they can be combined with many types of optimizers. We have seen O-ACCEL's efficiency with steepest descent, and accelerating quasi-Newton, Newton methods, and domain-specific methods has the potential to reduce costs for more expensive algorithms. For example, in the work of De Sterck,[1] it is shown that N-GMRES significantly accelerates the ALS algorithm, which already without acceleration performs much better than L-BFGS and N-CG on a standard canonical tensor decomposition problem. Our numerical tests show that O-ACCEL further improves the ALS convergence for this problem.

There are two particular paths of interest to improve the proposed acceleration scheme. The first is to reduce the cost by not using a line search between the proposed steps by the solver and O-ACCEL. One can instead rely on heuristics along the lines of those proposed by Washio et al.[2] The second is to find better heuristics for choosing previous iterates to use in the acceleration step. Currently, no choices are made, other than discarding all iterates when problems appear. Better guidelines for the number of previous iterates to store are another topic of interest, especially when memory storage is limited.

We would like to investigate connections between the proposed O-ACCEL acceleration step and other optimization procedures, in the same fashion that Fang et al.[6] put Anderson acceleration in the context of a family of Broyden-type approximations of the inverse Jacobian (Hessian). The preliminary analysis presented in this manuscript shows that, for convex quadratic objectives, O-ACCEL with a gradient descent preconditioner is equivalent to FOM for linear systems. As FOM is equivalent to CG for symmetric positive-definite systems, we can view O-ACCEL in the context of N-CG methods using a larger history size than usual. There are many new ideas for improving step directions based on previous iterates, such as the acceleration scheme by Scieur et al.[10] and Block BFGS by Gao et al.[23] A better understanding of the overlaps between these and more classical optimization procedures can provide useful guidance for further research.

Further work is needed to test O-ACCEL on a wider range of problems, with comparisons to other state-of-the-art implementations of solvers and accelerators, in order to provide guidance as to when a method is appropriate. For example, on Problems A–G, O-ACCEL accelerating steepest descent is superior to N-CG and slightly better than L-BFGS. These results may, however, be due to implementations from the work of De Sterck[1] and test problems favoring the acceleration algorithms. They are still indicative of the power of objective value-based optimization, a research track that is worth pursuing further.

ACKNOWLEDGEMENTS

The author would like to thank Coralia Cartis for her suggestions on how to present the results in this manuscript, and his supervisors Jeff Dewynne and Chris Farmer for their input. Antoine Levitt suggested to look into the connection between O-ACCEL and FOM. This work originally came about from an investigation of N-GMRES for nonlinear solvers for PDEs with Patrick Farrell (Riseth[24]). This publication is based on work partially supported by the EPSRC Centre for Doctoral Training in Industrially Focused Mathematical Modelling (EP/L015803/1).

DATA ACCESS

The code used in producing this manuscript is available at https://github.com/anriseth/objective_accel_code. It includes MATLAB implementations of the N-GMRES and O-ACCEL algorithms, and the code to run Problems A–G and the tensor optimization problem. Implementations of N-GMRES and O-ACCEL are also available in the optimization package Optim (Mogensen et al.[21]) of the Julia programming language (Bezanson et al.[22]).

ORCID

Asbjørn Nilsen Riseth http://orcid.org/0000-0002-5861-7885

REFERENCES

1. De Sterck H. Steepest descent preconditioning for nonlinear GMRES optimization. Numer Linear Algebra Appl. 2013;20(3):453–471.
2. Washio T, Oosterlee CW. Krylov subspace acceleration for nonlinear multigrid schemes. Electron Trans Numer Anal. 1997;6:271–290.
3. Oosterlee CW, Washio T. Krylov subspace acceleration of nonlinear multigrid with application to recirculating flows. SIAM J Sci Comput. 2000;21(5):1670–1690.
4. Anderson DG. Iterative procedures for nonlinear integral equations. J ACM. 1965;12(4):547–560.
5. Walker HF, Ni P. Anderson acceleration for fixed-point iterations. SIAM J Numer Anal. 2011;49(4):1715–1735.
6. Fang H, Saad Y. Two classes of multisecant methods for nonlinear acceleration. Numer Linear Algebra Appl. 2009;16(3):197–221.
7. Brune PR, Knepley MG, Smith BF, Tu X. Composing scalable nonlinear algebraic solvers. SIAM Review. 2015;57(4):535–565.
8. De Sterck H. A nonlinear GMRES optimization algorithm for canonical tensor decomposition. SIAM J Sci Comput. 2012;34(3): A1351–A1379.
9. De Sterck H, Howse A. Nonlinearly preconditioned optimization on Grassmann manifolds for computing approximate tucker tensor decompositions. SIAM J Sci Comput. 2016;38(2):A997–A1018.
10. Scieur D, d'Aspremont A, Bach F. Regularized nonlinear acceleration. In: Lee DD, Sugiyama M, Luxburg UV, Guyon I, Garnett R, editors. Advances in Neural Information Processing Systems 29 (NIPS 2016); 2016 Dec 5–10; Barcelona, Spain. Red Hook, NY: Curran Associates, Inc.; 2016. p. 712–720. Available from: http://papers.nips.cc/paper/6267-regularized-nonlinear-acceleration.pdf
11. Cartis C, Geleta M. Accelerating nonlinear optimization algorithms. Oxford, UK: University of Oxford; 2017.
12. Saad Y. Iterative methods for sparse linear systems. 2nd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics; 2003.
13. Nocedal J, Wright S. Numerical optimization. New York, NY: Science & Business Media; 2006.
14. Gould NIM, Orban D, Toint PL. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. Comput Optim Appl. 2015;60(3):545–557.
15. Dolan ED, Moré JJ. Benchmarking optimization software with performance profiles. Math Program. 2002;91(2):201–213.
16. Neumaier A. Solving ill-conditioned and singular linear systems: a tutorial on regularization. SIAM Rev. 1998;40(3):636–666.
17. Moré JJ, Garbow BS, Hillstrom KE. Testing unconstrained optimization software. ACM T Math Software. 1981;7(1):17–41.
18. Moré JJ, Thuente DJ. Line search algorithms with guaranteed sufficient decrease. ACM T Math Software. 1994;20(3):286–307.
19. Dunlavy DM, Kolda TG, Acar E. Poblano v1.0: a MATLAB toolbox for gradient-based optimization. Albuquerque, NM: Sandia National Laboratories; 2010. SAND2010-1422.
20. Acar E, Dunlavy DM, Kolda TG. A scalable optimization approach for fitting canonical tensor decompositions. J Chemom. 2011;25(2):67–86.
21. Mogensen PK, Riseth AN. Optim: a mathematical optimization package for Julia. J Open Source Softw. 2018;3(24):615.
22. Bezanson J, Edelman A, Karpinski S, Shah VB. Julia: a fresh approach to numerical computing. SIAM Review. 2017;59(1):65–98.
23. Gao W, Goldfarb D. Block BFGS methods. SIAM J Optim. 2018;28(2):1205–1231.
24. Riseth AN. Nonlinear solver techniques in reservoir management. Oxford, UK: University of Oxford; 2015. Technical Report. Available from: http://www.pefarrell.org/wp-content/uploads/2015/09/riseth2015.pdf

How to cite this article: Riseth AN. Objective acceleration for unconstrained optimization. *Numer Linear Algebra Appl*. 2019;26:e2216. https://doi.org/10.1002/nla.2216

APPENDIX

TABLES OF NUMERICAL RESULTS

To supplement the performance profiles in the manuscript, we include tables that present statistics of the solver performances for the individual test problems.

TABLE A1 Quantiles reporting f evaluations to reach tolerance for each solver on Problems A–D from Section 3.1

Algorithm	A, $n = 100$			A, $n = 200$		
	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$
O-ACCEL-B	75	79	81	103	107	111
O-ACCEL-A	131	136	140	171	179	184
N-CG	87	93	99	113	131	145
L-BFGS	75	79	81	103	107	111
N-GMRES-B	111	117	122	158	169	192
N-GMRES-A	166	246	336	307	414	510
Algorithm	**B, $n = 100$**			**B, $n = 200$**		
	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$
O-ACCEL-B	183	267	416	262	365	595
O-ACCEL-A	258	389	546	377	478	800
N-CG	134	211	560	221	359	1,598
L-BFGS	76	100	169	99	127	292
N-GMRES-B	215	315	542	317	433	840
N-GMRES-A	272	648	1,516	452	809	2,204
Algorithm	**C, $n = 100$**			**C, $n = 200$**		
	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$
O-ACCEL-B	113	136	178	151	176	214.5
O-ACCEL-A	188	208	259	264	292	324
N-CG	165	187	215	259	298	344
L-BFGS	104	114	125	148	160	177
N-GMRES-B	142	164	208	219	254	304
N-GMRES-A	264	333	459	508	620	854
Algorithm	**D, $n = 500$**			**D, $n = 1{,}000$**		
	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$
O-ACCEL-B	93	105	123	91	98	116
O-ACCEL-A	193	233	277	192	233	280
N-CG	158	188	196	162	190	197
L-BFGS	128	155	194	129	153	189
N-GMRES-B	141	163	193	142	167	193
N-GMRES-A	284	349	508	290	349	471
Algorithm	**D, $n = 50{,}000$**			**D, $n = 100{,}000$**		
	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$
O-ACCEL-B	101	117	132	122	126	135
O-ACCEL-A	195	226	276	196	225	271
N-CG	159	187	196	159	188	196
L-BFGS	131	156	190	130	156	191
N-GMRES-B	154	178	215	162.5	190	231
N-GMRES-A	312	378	525	320	394	562

Note. Gray rows highlight the solver with the best 0.5 quantile. L-BFGS performs best for the easier problems, whereas O-ACCEL handles the difficult problems better. In Problem A, the L-BFGS and O-ACCEL performance measures are so similar that the quantiles are the same. O-ACCEL = objective acceleration; N-CG = nonlinear conjugate gradient; L-BFGS = limited-memory Broyden–Fletcher–Goldfarb–Shanno; N-GMRES = nonlinear generalized minimal residual

Tables A1 and A2 show the results from test Problems A–G. Each of the problems was tested with different sizes n, and for each value n, the problems were run $1,000$ times in order to create statistics. The tables report the 0.1, 0.5, and 0.9 quantiles of f evaluations to reach the objective value reduction in (32), denoted by $Q_{0.1}$, $Q_{0.5}$, and $Q_{0.9}$, respectively. Table A1 provides results for Problems A–D, and Table A2 provides results for the remaining Problems E–G.

Tables A3 and A4 show the results from the CUTEst problems, where "Fail" means failure to reach the gradient value reduction in (36) within $2,000$ iterations. The norm used for the gradient values in the tables is the infinity norm.

TABLE A2 Quantiles reporting f evaluations to reach tolerance for each solver on Problems E–G from Section 3.1

Algorithm	E, $n = 100$			E, $n = 200$		
	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$
O-ACCEL-B	190	222	265	198	228	274
O-ACCEL-A	301	349	625	312	371	781
N-CG	205	238	283	213	245	290
L-BFGS	463	627	965	480	639	1,036
N-GMRES-B	232	267	330	235	268	338
N-GMRES-A	280	332	395	284	335	401
Algorithm	**E, $n = 50{,}000$**			**E, $n = 100{,}000$**		
	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$
O-ACCEL-B	368	487	689	400	536	798
O-ACCEL-A	745	1,157	1,356	764	1,207	1,417
N-CG	297	360	461	321	384	491
L-BFGS	599	703	852	626	725	879
N-GMRES-B	275	335	738	258	318	848
N-GMRES-A	310	391	562	318	402	609
Algorithm	**F, $n = 200$**			**F, $n = 500$**		
	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$
O-ACCEL-B	53	71	118	44	55	97
O-ACCEL-A	81	93	110	84	102	121
N-CG	34	46	60	33	47	69
L-BFGS	41	48	56	34	44	51
N-GMRES-B	48	59	110	43	51	89
N-GMRES-A	76	87	99	78	92	107
Algorithm	**G, $n = 100$**			**G, $n = 200$**		
	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$	$Q_{0.1}$	$Q_{0.5}$	$Q_{0.9}$
O-ACCEL-B	148	212	296	196	224	256
O-ACCEL-A	302	940	1,078	220	815	957
N-CG	76	191	201	53	165	174
L-BFGS	66	173	180	53	150	156
N-GMRES-B	161	216	266	167	210	245
N-GMRES-A	528	764	4,518	203	720	4,526

Note. Gray rows highlight the solver with the best 0.5 quantile. N-GMRES performs best at the median range for two problems; however, it is less robust as can be seen from the upper quantile. O-ACCEL = objective acceleration; N-CG = nonlinear conjugate gradient; L-BFGS = limited-memory Broyden–Fletcher–Goldfarb–Shanno; N-GMRES = nonlinear generalized minimal residual

TABLE A3 Results from the CUTEst problems

Problem		Solver	Iter	f-calls	$f_{\min}$	$\|g_{\min}\|$	Fail
	$n = 5{,}000$	O-ACCEL-B	9	20	0.0	2.3×10^{-6}	
ARWHEAD	$f_0 = 1.5 \times 10^4$	L-BFGS	6	21	0.0	4.8×10^{-7}	
	$\|g_0\| = 4.0 \times 10^4$	N-GMRES-B	6	20	2.6×10^{-9}	2.1×10^{-5}	
	$n = 10{,}000$	O-ACCEL-B	19	86	-1.9×10^3	2.0×10^{-10}	
BOX	$f_0 = 0.0$	L-BFGS	7	14	-1.9×10^3	4.0×10^{-9}	
	$\|g_0\| = 5.0 \times 10^{-1}$	N-GMRES-B	30	105	-1.9×10^3	2.2×10^{-9}	
	$n = 10{,}000$	O-ACCEL-B	52	272	-1.0×10^4	7.1×10^{-9}	
COSINE	$f_0 = 8.8 \times 10^3$	L-BFGS	12	22	-1.0×10^4	3.1×10^{-9}	
	$\|g_0\| = 9.6 \times 10^{-1}$	N-GMRES-B	29	80	-1.0×10^4	2.0×10^{-9}	
	$n = 5{,}000$	O-ACCEL-B	157	478	1.7×10^3	4.7×10^{-5}	
CRAGGLVY	$f_0 = 2.7 \times 10^6$	L-BFGS	57	133	1.7×10^3	4.0×10^{-5}	
	$\|g_0\| = 5.6 \times 10^3$	N-GMRES-B	229	760	1.7×10^3	3.1×10^{-5}	
	$n = 3{,}000$	O-ACCEL-B	48	223	1.0	6.1×10^{-10}	
DIXMAANA	$f_0 = 2.9 \times 10^4$	L-BFGS	6	12	1.0	9.9×10^{-16}	
	$\|g_0\| = 2.8 \times 10^1$	N-GMRES-B	13	53	1.0	1.5×10^{-10}	
	$n = 3{,}000$	O-ACCEL-B	31	99	1.0	3.2×10^{-8}	
DIXMAANB	$f_0 = 4.7 \times 10^4$	L-BFGS	6	11	1.0	2.3×10^{-7}	
	$\|g_0\| = 4.0 \times 10^1$	N-GMRES-B	35	228	1.0	3.8×10^{-10}	
	$n = 3{,}000$	O-ACCEL-B	30	105	1.0	4.1×10^{-8}	
DIXMAANC	$f_0 = 8.2 \times 10^4$	L-BFGS	7	14	1.0	2.3×10^{-8}	
	$\|g_0\| = 7.6 \times 10^1$	N-GMRES-B	13	49	1.0	4.4×10^{-8}	
	$n = 3{,}000$	O-ACCEL-B	23	75	1.0	1.5×10^{-6}	
DIXMAAND	$f_0 = 1.6 \times 10^5$	L-BFGS	8	16	1.0	2.6×10^{-7}	
	$\|g_0\| = 1.5 \times 10^2$	N-GMRES-B	16	78	1.0	4.9×10^{-7}	
	$n = 3{,}000$	O-ACCEL-B	394	1,109	1.0	2.6×10^{-7}	
DIXMAANE	$f_0 = 2.2 \times 10^4$	L-BFGS	241	485	1.0	2.7×10^{-7}	
	$\|g_0\| = 2.7 \times 10^1$	N-GMRES-B	885	2,751	1.0	2.6×10^{-7}	
	$n = 3{,}000$	O-ACCEL-B	282	765	1.0	3.5×10^{-7}	
DIXMAANF	$f_0 = 4.1 \times 10^4$	L-BFGS	195	393	1.0	3.7×10^{-7}	
	$\|g_0\| = 3.9 \times 10^1$	N-GMRES-B	567	1,687	1.0	3.9×10^{-7}	
	$n = 3{,}000$	O-ACCEL-B	277	770	1.0	6.9×10^{-7}	
DIXMAANG	$f_0 = 7.6 \times 10^4$	L-BFGS	165	334	1.0	6.9×10^{-7}	
	$\|g_0\| = 7.5 \times 10^1$	N-GMRES-B	570	1,673	1.0	7.2×10^{-7}	
	$n = 3{,}000$	O-ACCEL-B	236	655	1.0	1.5×10^{-6}	
DIXMAANH	$f_0 = 1.5 \times 10^5$	L-BFGS	146	297	1.0	1.5×10^{-6}	
	$\|g_0\| = 1.5 \times 10^2$	N-GMRES-B	620	1,835	1.0	1.4×10^{-6}	
	$n = 3{,}000$	O-ACCEL-B	862	1,877	1.0	7.1×10^{-7}	
DIXMAANK	$f_0 = 7.4 \times 10^4$	L-BFGS	392	787	1.0	7.1×10^{-7}	
	$\|g_0\| = 7.4 \times 10^1$	N-GMRES-B	556	1,640	1.0	7.2×10^{-7}	
	$n = 3{,}000$	O-ACCEL-B	267	685	1.0	1.4×10^{-6}	
DIXMAANL	$f_0 = 1.5 \times 10^5$	L-BFGS	240	485	1.0	1.5×10^{-6}	
	$\|g_0\| = 1.5 \times 10^2$	N-GMRES-B	378	1,096	1.0	1.4×10^{-6}	
	$n = 3{,}000$	O-ACCEL-B	1,426	3,149	1.0	1.3×10^{-6}	
DIXMAANP	$f_0 = 7.1 \times 10^4$	L-BFGS	549	1,102	1.0	1.3×10^{-6}	
	$\|g_0\| = 1.3 \times 10^2$	N-GMRES-B	1,037	3,091	1.0	1.1×10^{-6}	
	$n = 2{,}000$	O-ACCEL-B	75	246	1.20×10^4	1.8×10^{-5}	
EDENSCH	$f_0 = 7.4 \times 10^6$	L-BFGS	21	45	1.20×10^4	2.0×10^{-5}	
	$\|g_0\| = 2.2 \times 10^3$	N-GMRES-B	54	200	1.20×10^4	7.1×10^{-6}	

Note. O-ACCEL = objective acceleration; L-BFGS = limited-memory Broyden–Fletcher–Goldfarb–Shanno; N-GMRES = nonlinear generalized minimal residual

TABLE A4 Results from the CUTEst tests

Problem		Solver	Iter	f-calls	$f_{\min}$	$\|g_{\min}\|$	Fail
	$n = 1{,}000$	O-ACCEL-B	6	14	-10.0×10^{2}	1.7×10^{-6}	
EG2	$f_0 = -8.4 \times 10^{2}$	L-BFGS	3	9	-10.0×10^{2}	4.1×10^{-7}	
	$\|g_0\| = 5.4 \times 10^{2}$	N-GMRES-B	6	14	-10.0×10^{2}	3.6×10^{-6}	
	$n = 5{,}000$	O-ACCEL-B	47	250	5.6×10^{3}	8.5×10^{-7}	
ENGVAL1	$f_0 = 2.9 \times 10^{5}$	L-BFGS	33	366	5.6×10^{3}	2.8×10^{-6}	×
	$\|g_0\| = 1.2 \times 10^{2}$	N-GMRES-B	65	318	5.6×10^{3}	7.6×10^{-7}	
	$n = 5{,}000$	O-ACCEL-B	143	989	-2.5×10^{5}	4.8×10^{-9}	
FLETBV3M	$f_0 = 2.0 \times 10^{2}$	L-BFGS	24	62	-2.5×10^{5}	5.1×10^{-10}	
	$\|g_0\| = 7.1 \times 10^{-1}$	N-GMRES-B	92	756	-2.5×10^{5}	5.0×10^{-9}	
	$n = 5{,}625$	O-ACCEL-B	1,510	4,384	1.0	9.6×10^{-9}	×
FMINSRF2	$f_0 = 2.8 \times 10^{1}$	L-BFGS	1,537	3,367	1.0	2.2×10^{-10}	
	$\|g_0\| = 2.4 \times 10^{-2}$	N-GMRES-B	2,000	4,661	1.5	9.1×10^{-3}	×
	$n = 5{,}625$	O-ACCEL-B	1,741	5,007	1.0	9.7×10^{-9}	×
FMINSURF	$f_0 = 2.9 \times 10^{1}$	L-BFGS	781	1,672	1.0	2.2×10^{-10}	
	$\|g_0\| = 2.3 \times 10^{-2}$	N-GMRES-B	2,000	4,440	1.6	9.2×10^{-3}	×
	$n = 5{,}010$	O-ACCEL-B	526	1,610	-1.1×10^{3}	3.1×10^{-8}	
NCB20	$f_0 = 1.0 \times 10^{4}$	L-BFGS	467	1,089	-1.2×10^{3}	1.3×10^{-6}	×
	$\|g_0\| = 4.0$	N-GMRES-B	2,000	4,093	-1.1×10^{3}	6.8×10^{-1}	×
	$n = 5{,}000$	O-ACCEL-B	1,178	4,121	7.4×10^{3}	3.3×10^{-8}	
NCB20B	$f_0 = 1.0 \times 10^{4}$	L-BFGS	1,467	4,173	7.4×10^{3}	3.0×10^{-5}	×
	$\|g_0\| = 4.0$	N-GMRES-B	2,000	3,013	7.4×10^{3}	2.7×10^{-5}	×
	$n = 5{,}000$	O-ACCEL-B	363	1,044	1.9×10^{-4}	1.7×10^{-4}	
NONDQUAR	$f_0 = 5.0 \times 10^{3}$	L-BFGS	208	436	9.3×10^{-5}	1.8×10^{-4}	
	$\|g_0\| = 2.0 \times 10^{4}$	N-GMRES-B	480	1,394	3.8×10^{-4}	1.7×10^{-4}	
	$n = 200$	O-ACCEL-B	283	1,798	1.0×10^{-3}	1.5×10^{-3}	
PENALTY3	$f_0 = 1.6 \times 10^{9}$	L-BFGS	3	15	1.6×10^{9}	1.6×10^{5}	×
	$\|g_0\| = 1.6 \times 10^{5}$	N-GMRES-B	2,000	2,165	2.4×10^{172}	2.4×10^{172}	×
	$n = 5{,}000$	O-ACCEL-B	115	460	3.3×10^{-6}	1.2×10^{-6}	
POWELLSG	$f_0 = 2.7 \times 10^{5}$	L-BFGS	20	49	4.2×10^{-10}	5.5×10^{-7}	
	$\|g_0\| = 3.1 \times 10^{2}$	N-GMRES-B	105	308	3.2×10^{-9}	1.4×10^{-8}	
	$n = 10{,}000$	O-ACCEL-B	186	637	4.0×10^{4}	1.5×10^{4}	
POWER	$f_0 = 2.5 \times 10^{15}$	L-BFGS	38	107	4.9×10^{4}	1.5×10^{4}	
	$\|g_0\| = 2.0 \times 10^{12}$	N-GMRES-B	607	1,868	7.5×10^{4}	1.4×10^{4}	
	$n = 5{,}000$	O-ACCEL-B	74	208	-1.5×10^{4}	1.0×10^{-8}	
SCHMVETT	$f_0 = -1.4 \times 10^{4}$	L-BFGS	55	133	-1.5×10^{4}	9.2×10^{-9}	
	$\|g_0\| = 1.1$	N-GMRES-B	78	239	-1.5×10^{4}	9.3×10^{-9}	
	$n = 5{,}000$	O-ACCEL-B	78	297	-6.8×10^{6}	4.9×10^{-5}	
SINQUAD	$f_0 = 6.6 \times 10^{-1}$	L-BFGS	12	45	-6.8×10^{6}	1.4×10^{-5}	
	$\|g_0\| = 5.0 \times 10^{3}$	N-GMRES-B	95	483	-6.8×10^{6}	7.1×10^{-6}	
	$n = 10{,}000$	O-ACCEL-B	134	598	1.7×10^{-7}	4.4×10^{-6}	
SPARSQUR	$f_0 = 1.4 \times 10^{7}$	L-BFGS	26	91	1.4×10^{-6}	2.3×10^{-4}	
	$\|g_0\| = 3.2 \times 10^{4}$	N-GMRES-B	214	797	1.1×10^{-6}	1.8×10^{-5}	
	$n = 50$	O-ACCEL-B	201	538	1.4×10^{3}	1.3×10^{-6}	
TOINTGOR	$f_0 = 5.1 \times 10^{3}$	L-BFGS	126	265	1.4×10^{3}	9.8×10^{-7}	
	$\|g_0\| = 1.5 \times 10^{2}$	N-GMRES-B	287	795	1.4×10^{3}	1.3×10^{-6}	
	$n = 50$	O-ACCEL-B	277	963	2.3×10^{2}	6.3×10^{-8}	
TOINTPSP	$f_0 = 1.8 \times 10^{3}$	L-BFGS	127	334	2.3×10^{2}	2.9×10^{-7}	
	$\|g_0\| = 3.0 \times 10^{1}$	N-GMRES-B	423	1,287	2.3×10^{2}	2.5×10^{-7}	
	$n = 200$	O-ACCEL-B	8	28	6.9×10^{-2}	1.2×10^{2}	
VARDIM	$f_0 = 3.3 \times 10^{16}$	L-BFGS	1	20	3.3×10^{16}	1.9×10^{15}	×
	$\|g_0\| = 1.9 \times 10^{15}$	N-GMRES-B	4	39	5.3×10^{3}	5.0×10^{5}	

Note. O-ACCEL = objective acceleration; L-BFGS = limited-memory Broyden–Fletcher–Goldfarb–Shanno; N-GMRES = nonlinear generalized minimal residual