# Nonlinear conjugate gradient for smooth convex functions[*]

Sahar Karimi[†] Stephen Vavasis[‡]

September 4, 2021

**Abstract**

The method of nonlinear conjugate gradients (NCG) is widely used in practice for unconstrained optimization, but it satisfies weak complexity bounds at best when applied to smooth convex functions. In contrast, Nesterov's accelerated gradient (AG) method is optimal up to constant factors for this class.

However, when specialized to quadratic function, conjugate gradient is optimal in a strong sense among function-gradient methods. Therefore, there is seemingly a gap in the menu of available algorithms: NCG, the optimal algorithm for quadratic functions that also exhibits good practical performance for general functions, has poor complexity bounds compared to AG.

We propose an NCG method called C+AG ("conjugate plus accelerated gradient") to close this gap, that is, it is optimal for quadratic functions and still satisfies the best possible complexity bound for more general smooth convex functions. It takes conjugate gradient steps until insufficient progress is made, at which time it switches to accelerated gradient steps, and later retries conjugate gradient. The proposed method has the following theoretical properties: (i) It is identical to linear conjugate gradient (and hence terminates finitely) if the objective function is quadratic; (ii) Its running-time bound is $O(\epsilon^{-1/2})$ gradient evaluations for an $L$-smooth convex function, where $\epsilon$ is the desired residual reduction, (iii) Its running-time bound is $O(\sqrt{L/\ell}\ln(1/\epsilon))$ if the function is both $L$-smooth and $\ell$-strongly convex. We also conjecture and outline a proof that a variant of the method has the property: (iv) It is $n$-step quadratically convergent for a function whose second derivative is smooth and invertible at the optimizer. Note that the bounds in (ii) and (iii) match AG and are the best possible, i.e., they match lower bounds up to constant factors for the classes of functions under consideration. On the other hand, (i) and (iv) match NCG.

[*]Supported in part by a grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

[†]Department of Combinatorics & Optimization, University of Waterloo, 200 University Ave. W., Waterloo, ON, N2L 3G1, Canada, sahar.karimi@gmail.com.

[‡]Department of Combinatorics & Optimization, University of Waterloo, 200 University Ave. W., Waterloo, ON, N2L 3G1, Canada, vavasis@uwaterloo.ca.

In computational tests, the function-gradient evaluation count for the C+AG method typically behaves as whichever is better of AG or classical NCG. In most test cases it outperforms both.

## 1 First-order methods for smooth convex functions

The problem under consideration is minimizing an unconstrained $L$-smooth convex function $f : \mathbb{R}^n \to \mathbb{R}$. Recall that a convex function is *$L$-smooth* if it is differentiable and for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$,

$$f(\boldsymbol{y}) - f(\boldsymbol{x}) - \nabla f(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x}) \le L\|\boldsymbol{x} - \boldsymbol{y}\|^2/2. \tag{1}$$

Note that convexity implies that the quantity on the left-hand side of this inequality is nonnegative. We will also sometimes consider *$\ell$-strongly convex* functions, which are defined to be those functions satisfying

$$f(\boldsymbol{y}) - f(\boldsymbol{x}) - \nabla f(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x}) \ge \ell\|\boldsymbol{x} - \boldsymbol{y}\|^2/2 \tag{2}$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ for some modulus $\ell \ge 0$. Note that $\ell = 0$ means simply that $f$ is differentiable and convex.

Nonlinear conjugate gradient (NCG) [16] is perhaps the most widely used first-order method for unconstrained optimization. It is an extension of linear conjugate gradient (LCG) introduced by Hestenes and Stiefel [8] to minimize convex quadratic functions. The smoothness modulus $L$ for a quadratic function $f(\boldsymbol{x}) = \boldsymbol{x}^T A\boldsymbol{x}/2 - \boldsymbol{b}^T\boldsymbol{x} + c$, where $A \in \mathbb{R}^{n\times n}$ is positive semidefinite, is the maximum eigenvalue of $A$. In fact, LCG is optimal for quadratic functions in a strong sense that $f(\boldsymbol{x}_k)$, where $k$ is the $k$th iterate, is minimized over all choices of $\boldsymbol{x}_k$ lying in the $k$th Krylov space. Daniel [4] proved that conjugate gradient requires $k = O(\ln(1/\epsilon)\sqrt{L/\ell})$ iterations to reduce the residual of $f$ by a factor $\epsilon$, i.e., so that $(f(\boldsymbol{x}_k) - f^*)/(f(\boldsymbol{x}_0) - f^*) \le \epsilon$. Here and in the remainder of the paper, $f^*$ stands for $\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x})$ and is assumed to be attained.

However, if we enlarge the class of problems to include all smooth, strongly convex functions, then NCG fails to satisfy this complexity bound and could perform even worse than steepest descent as shown by Nemirovsky and Yudin [13]. Nonetheless, in practice, nonlinear conjugate gradient is difficult to beat even by algorithms with better theoretical guarantees, as observed in our experiments and by others, e.g., Carmon et al. [2].

Several first-order methods have been proposed for smooth strongly convex functions that achieve the running time $O(\ln(1/\epsilon)\sqrt{L/\ell})$ in the including Nemirovsky and Yudin's [13] method, which requires solution of a two-dimensional subproblem on each iteration, Nesterov's accelerated gradient (AG) [14], and Geometric Descent (GD) [1]. In the case of GD, the bound is on iterations rather than gradient evaluations since each iteration requires additional gradient evaluations for the line search. It is known that $\Omega(\ln(1/\epsilon)\sqrt{L/\ell})$ gradient evaluations are required to reduce the objective function residual by a factor of $\epsilon$; see, e.g., the book of Nesterov [15] for a discussion of the lower bound result. Since the algorithms mentioned in this paragraph match the lower bound, these algorithms are optimal up to a constant factor. In the case of smooth convex (not strongly convex) functions, AG requires $O(\epsilon^{-1/2})$ iterations.

Table 1: Minimization of $f(\boldsymbol{x}) = \boldsymbol{x}^T A\boldsymbol{x}/2 - \boldsymbol{b}^T\boldsymbol{x}$, where $A = \mathrm{Diag}(1, 4, 9, \ldots, 1000^2)$ and $\boldsymbol{b} = [\sin 1, \sin 2, \ldots, \sin 1000]$, with four algorithms to a tolerance of $\|\nabla f(\boldsymbol{x})\| \le 10^{-6}$

| | LCG | C+AG | NCG | AG |
|---|---|---|---|---|
| # iterations | 1503 | 1505 | 1508 | 17816 |
| # function/gradient evaluations | 1503 | 3017 | 3015 | 17816 |

A technical issue is as follows. In the bounds for AG in the previous paragraph, $\epsilon$ stands for $(f(\boldsymbol{x}_k) - f^*)/(L\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|^2)$, where $\boldsymbol{x}^*$ is some optimizer. On the other hand, in the case of LCG, $\epsilon$ stands for $(f(\boldsymbol{x}_k) - f^*)/(f(\boldsymbol{x}_0) - f^*)$. The latter $\epsilon$ is always greater than or equal to the former, a consequence of a consequence of $L$-smoothness. See [15] Therefore, the LCG bound is slightly better than the corresponding AG bound in this technical sense. We will use the former interpretation of $\epsilon$ for the remainder of the paper in order to obtain bounds applying to both CG and AG.

In practice, conjugate gradient for quadratic functions is typically superior to AG applied to quadratic functions. The reason appears to be not so much the technical reason mentioned in the previous paragraph but the fact that LCG is optimal in a stronger sense than AG: the iterate $\boldsymbol{x}_k$ is the true minimizer of $f(\boldsymbol{x})$ over the $k$th Krylov space, which is defined to be $\{\boldsymbol{b}, A\boldsymbol{b}, \ldots, A^{k-1}\boldsymbol{b}\}$ assuming $\boldsymbol{x}_0 = \boldsymbol{0}$. For example, consider solving the linear system $A\boldsymbol{x} = \boldsymbol{b}$ or (equivalently) minimizing $\boldsymbol{x}^T A\boldsymbol{x}/2 - \boldsymbol{b}^T\boldsymbol{x}$ where $A = \mathrm{Diag}(1, 4, 9, \ldots, 1000^2)$ and $\boldsymbol{b} = [\sin 1, \sin 2, \ldots, \sin 1000]^T$. The problem can be solved with LCG, NCG, or AG. Table 1 shows the results for these algorithms. (Of course, the problem can also be solved by a trivial algorithm because we chose a diagonal coefficient matrix, but none of the algorithms under consideration use this fact.)

Note that the three conjugate gradients require roughly the same number of iterations, all much less than AG. The function evaluations are: 1 per LCG iteration (i.e., a single matrix-vector multiplication), 2 per C+AG iteration, approximately 2 per NCG iteration, and 1 per AG iteration. The requirement for more than one function evaluation per NCG iteration stems from the need to select a step-size. We return to this point below.

In Table 1 and for the remainder of this paper, we speak of a "function-gradient evaluation", which means, for a given $\boldsymbol{x}$, the simultaneous evaluation of $f(\boldsymbol{x})$ and $\nabla f(\boldsymbol{x})$. We regard function-gradient evaluation as the atom of work. This metric may be inaccurate in the setting that $f(\boldsymbol{x})$ can be evaluated much more efficiently than $\nabla f(\boldsymbol{x})$. However, in most applications evaluating $f, \nabla f$ together is not much more expensive than evaluating either separately.

It is also known that NCG is fast for functions that are nearly quadratic. In particular, a classic result of Cohen [3] shows that NCG exhibits $n$-step quadratic convergence near the minimizer provided that the objective function is nearly quadratic in this neighborhood.

Thus, there is seemingly a gap in the menu of available algorithms because NCG, though optimal for quadratic functions and fast for nearly quadratic functions, is not able to achieve reasonable complexity bounds for the larger class of smooth, strongly convex functions. On the other hand, the methods with the good complexity bounds for this class are all suboptimal for quadratic functions.

We propose C+AG to close this gap. The method is based on the following ideas:

1. A line-search for NCG described in Section 2 is used that is exact for quadratic functions but could be poor for general functions. Failure of the line-search is caught by the progress measure described in the next item.

2. A progress measure described in Section 3 for nonlinear conjugate gradient based on Nesterov's estimating sequence is checked. The progress measure is inexpensive to evaluate on every iteration. It requires prior knowledge of $L, \ell$.

3. On iterations when insufficient progress is made, C+AG switches to AG to guarantee reduction in the progress measure as explained in Section 4.

4. C+AG switches from AG back to NCG after a fixed reduction in the gradient norm. When CG resumes, the progress measure needs a modification as discussed in Section 5.

When applied to a quadratic function, the progress measure is always satisfied (this is proved in Section 3), and hence all steps will be NCG (and therefore LCG) steps. The fact that the method satisfies the $O(\ln(1/\epsilon)\sqrt{L/\ell})$ running-time bound of AG follows because of the progress measure. The conjectured final theoretical property of the method, namely, $n$-step quadratic convergence, is described in Section 6. Computational testing in Section 8 indicates that the running time of the method measured in function-gradient evaluations is roughly equal to whichever of AG or NCG is better for the problem at hand, but in most cases it outperforms both of them. We remark that C+AG requires prior knowledge of $L, \ell$; we return to this point in Section 9. In the case of a smooth convex function that is not strongly convex, we take $\ell = 0$.

We conclude this introductory section with a few remarks about our previous unpublished manuscript [11]. That work explored connections between AG, GD, and LCG, and proposed a hybrid of NCG and GD. However, that work did not address the number of function-gradient evaluations because the the evaluation count for the line search was not analyzed. Closer to what is proposed here is a hybrid algorithm from the first author's PhD thesis [10]. Compared to the PhD thesis, the proposed algorithm herein has several improvements including the line search and the method for switching from AG to CG as well as additional analysis.

## 2 NCG and line search

The full C+AG algorithm is presented in Section 7. In this section, we present the NCG portion of C+AG, which is identical to most other NCG routines and is as follows

```
$\boldsymbol{x}_0 := \mathbf{0}$;
$\boldsymbol{p}_1 := -\nabla f(\boldsymbol{x}_0)$;
$k := 0$;
```

```
while $\|\nabla g(\boldsymbol{x}_k)\| >$ tol
    Select $\alpha_{k+1}$;
    $\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \alpha_{k+1}\boldsymbol{p}_{k+1}$;
    Confirm that $\boldsymbol{x}_{k+1}$ satisfies the progress measure.
    Select $\beta_{k+1}$;
    $\boldsymbol{p}_{k+2} := -\nabla f(\boldsymbol{x}_{k+1}) + \beta_{k+1}\boldsymbol{p}_{k+1}$;
    $k := k + 1$;
```

For $\beta_{k+1}$ we use the Hager-Zhang formula [7], which is as follows:

$$\begin{aligned}
\hat{\boldsymbol{y}} &:= \boldsymbol{g}_{k+1} - \boldsymbol{g}_k; \\
\beta^1 &:= \left(\hat{\boldsymbol{y}} - \boldsymbol{p}_{k+1} \cdot \frac{2\|\hat{\boldsymbol{y}}\|^2}{\hat{\boldsymbol{y}}^T \boldsymbol{p}_{k+1}}\right)^T \frac{\boldsymbol{g}_{k+1}}{\hat{\boldsymbol{y}}^T \boldsymbol{p}_{k+1}}, \\
\beta^2 &:= \frac{-1}{\|\boldsymbol{p}_{k+1}\| \cdot \min\left(.01\|\boldsymbol{g}_0\|, \|\boldsymbol{g}_{k+1}\|\right)}. \\
\beta_{k+1} &:= \max(\beta^1, \beta^2).
\end{aligned}$$

For $\alpha_{k+1}$, we use the following formulas.

$$\tilde{\boldsymbol{x}} := \boldsymbol{x}_k + \boldsymbol{p}_{k+1}/L, \tag{3}$$

$$\boldsymbol{s} := L(\nabla f(\tilde{\boldsymbol{x}}) - \nabla f(\boldsymbol{x}_k)), \tag{4}$$

$$\alpha_{k+1} := \frac{-\nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_{k+1}}{\boldsymbol{p}_{k+1}^T \boldsymbol{s}}. \tag{5}$$

The rationale for this choice of $\alpha_{k+1}$ is as follows. In the case that $f(\boldsymbol{x}) = \boldsymbol{x}^T A\boldsymbol{x}/2 - \boldsymbol{b}^T\boldsymbol{x} + c$ (quadratic), $\nabla f(\boldsymbol{x}_k) = A\boldsymbol{x}_k - \boldsymbol{b}$ and $\nabla f(\tilde{\boldsymbol{x}}) = A\tilde{\boldsymbol{x}} - \boldsymbol{b} = A\boldsymbol{x}_k - \boldsymbol{b} + A\boldsymbol{p}_{k+1}/L$, and therefore $\boldsymbol{s} = A\boldsymbol{p}_{k+1}$ and $\boldsymbol{p}_{k+1}^T\boldsymbol{s} = \boldsymbol{p}_{k+1}^T A\boldsymbol{p}_{k+1}$. In this case, $\alpha_{k+1}$ is the exact minimizer of the univariate quadratic function $\alpha \mapsto f(\boldsymbol{x}_k + \alpha\boldsymbol{p}_{k+1})$.

In the case of a nonquadratic function, this choice of $\alpha_{k+1}$ could be inaccurate, which could lead to a poor choice for $\boldsymbol{x}_{k+1}$. The progress measure in the next section will catch this failure and select a new search direction $\boldsymbol{p}_{k+1}$ in this case.

## 3 Progress measure

The progress measure is taken from Nesterov's [15] analysis of the AG method. The basis of the progress measure is a sequence of strongly convex quadratic functions $\phi_0(\boldsymbol{x}), \phi_1(\boldsymbol{x}), \ldots$ called an "estimate sequence" in [15].

The sequence is initialized with

$$\phi_0(\boldsymbol{x}) := f(\boldsymbol{x}_0) + \frac{L}{2}\|\boldsymbol{x} - \boldsymbol{x}_0\|^2. \tag{6}$$

Then $\phi_k$ for $k \geq 1$ is defined in terms of its predecessor via:

$$\phi_{k+1}(\boldsymbol{x}) = (1 - \theta_k)\phi_k(\boldsymbol{x}) + \theta_k \left[ f(\bar{\boldsymbol{x}}_k) + \nabla f(\bar{\boldsymbol{x}}_k)^T(\boldsymbol{x} - \bar{\boldsymbol{x}}_k) + \frac{\ell}{2}\|\boldsymbol{x} - \bar{\boldsymbol{x}}_k\|^2 \right]. \tag{7}$$

Here, $\theta_k \in (0,1)$ and $\bar{\boldsymbol{x}}_k \in \mathbb{R}^n$ are detailed below. Note that the Hessian of $\phi_0$ and of the square-bracketed quadratic function in (7) are both multiples of $I$, and therefore inductively the Hessian of $\phi_k$ is a multiple of $I$ for all $k$. In other words, for each $k$ there exists $\gamma_k, \boldsymbol{v}_k, \phi_k^*$ such that $\phi_k(\boldsymbol{x}) = \phi_k^* + \gamma_k\|\boldsymbol{x} - \boldsymbol{v}_k\|^2$, where $\gamma_k > 0$. The formulas for $\gamma_{k+1}$, $\boldsymbol{v}_{k+1}$, $\phi_{k+1}^*$ are straightforward to obtain from (7) given $\gamma_k$, $\boldsymbol{v}_k$, $\phi_k^*$ as well as $\theta_k$ and $\bar{\boldsymbol{x}}_k$. These formulas appear in [15, p. 73] and are repeated here for the sake of completeness:

$$\gamma_{k+1} := (1-\theta_k)\gamma_k + \theta_k \ell, \tag{8}$$

$$\boldsymbol{v}_{k+1} := \frac{1}{\gamma_{k+1}}[(1-\theta_k)\gamma_k \boldsymbol{v}_k + \theta_k \ell \bar{\boldsymbol{x}}_k - \theta_k \nabla f(\bar{\boldsymbol{x}}_k)], \tag{9}$$

$$\begin{aligned} \phi_{k+1}^* := {} & (1-\theta_k)\phi_k^* + \theta_k f(\bar{\boldsymbol{x}}_k) - \frac{\theta_k^2}{2\gamma_{k+1}}\|\nabla f(\bar{\boldsymbol{x}}_k)\|^2 \\ & + \frac{\theta_k(1-\theta_k)\gamma_k}{\gamma_{k+1}}(\ell\|\bar{\boldsymbol{x}}_k - \boldsymbol{v}_k\|^2/2 + \nabla f(\bar{\boldsymbol{x}}_k)^T(\boldsymbol{v}_k - \bar{\boldsymbol{x}}_k)). \end{aligned} \tag{10}$$

To complete the formulation of $\phi_k(\cdot)$, we now specify $\theta_k$ and $\bar{\boldsymbol{x}}_k$. Scalar $\theta_k$ is selected as the positive root of the quadratic equation

$$L\theta_k^2 + (\gamma_k - \ell)\theta_k - \gamma_k = 0. \tag{11}$$

The fact that $\theta_k \in (0,1)$ is easily seen by observing the sign-change in this quadratic over $[0,1]$. A consequence of (11) and (8) is the identity

$$\frac{\theta_k^2}{2\gamma_{k+1}} = \frac{1}{2L}. \tag{12}$$

Finally, C+AG has three cases for selecting $\bar{\boldsymbol{x}}_k$. The first case is $\bar{\boldsymbol{x}}_k := \boldsymbol{x}_k$. The second and third cases are presented later.

The principal theorem assuring progress is Theorem 2.2.2 of Nesterov [15], which we restate here for the sake of completeness. The validity of this theorem does not depend on how the sequence $\bar{\boldsymbol{x}}_k$ is selected in the definition of $\phi_k(\cdot)$.

**Theorem 1** *(Nesterov) Suppose that for each $k = 0, 1, 2, 3, \ldots$, the iterate $\boldsymbol{x}_k$ satisfies $f(\boldsymbol{x}_k) \le \min_{\boldsymbol{x}} \phi_k(\boldsymbol{x}) \equiv \phi_k^*$. Then*

$$f(\boldsymbol{x}_k) - f^* \le L \min\left(\left(1 - \sqrt{\ell/L}\right)^k, \frac{4}{(k+2)^2}\right)\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|^2. \tag{13}$$

The sufficient-progress test used for CG is the condition of the theorem, namely, $f(\boldsymbol{x}_k) \le \phi_k^*$. As long as this test holds, the algorithm continues to take CG steps, and the right-hand side of (13) assures us that the optimal complexity bound is achieved (up to constants) in the case of both smooth convex functions and smooth, strongly convex functions.

The main result of this section is the following theorem, which states that if the objective function is quadratic, then the progress measure is always satisfied by conjugate gradient.

**Theorem 2** *If $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$, where $A$ is positive definite, and if $\bar{\boldsymbol{x}}_k$ is defined as $\bar{\boldsymbol{x}}_k := \boldsymbol{x}_k$, then $f(\boldsymbol{x}_k) \le \phi_k^*$ for every iteration $k$, where $\boldsymbol{x}_k$ is the sequence of conjugate gradient iterates.*

**Proof.** The result clearly holds when $k = 0$ by (6). Assuming $f(\boldsymbol{x}_k) \le \phi_k^*$, we now show that $f(\boldsymbol{x}_{k+1}) \le \phi_{k+1}^*$ For $k \ge 0$,

$$\begin{aligned}
\phi_{k+1}^* &= (1-\theta_k)\phi_k^* + \theta_k f(\boldsymbol{x}_k) - \frac{\theta_k^2}{2\gamma_{k+1}}\|\nabla f(\boldsymbol{x}_k)\|^2 \\
&\quad + \frac{\theta_k(1-\theta_k)\gamma_k}{\gamma_{k+1}}\left(\ell\|\boldsymbol{x}_k - \boldsymbol{v}_k\|^2 + \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{v}_k - \boldsymbol{x}_k)\right) &(14)\\
&\ge f(\boldsymbol{x}_k) - \frac{\theta_k^2}{2\gamma_{k+1}}\|\nabla f(\boldsymbol{x}_k)\|^2 + \frac{\theta_k(1-\theta_k)\gamma_k}{\gamma_{k+1}}\left(\nabla f(\boldsymbol{x}_k)^T(\boldsymbol{v}_k - \boldsymbol{x}_k)\right) &(15)\\
&= f(\boldsymbol{x}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{x}_k)\|^2 + \frac{\theta_k(1-\theta_k)\gamma_k}{\gamma_{k+1}}\left(\nabla f(\boldsymbol{x}_k)^T(\boldsymbol{v}_k - \boldsymbol{x}_k)\right). &(16)\\
&= f(\boldsymbol{x}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{x}_k)\|^2 &(17)\\
&\ge f(\boldsymbol{x}_k - \nabla f(\boldsymbol{x}_k)/L) &(18)\\
&\ge f(\boldsymbol{x}_{k+1}). &(19)
\end{aligned}$$

Here, (14) is a restatement of (10) under the assumption that $\bar{\boldsymbol{x}}_k = \boldsymbol{x}_k$. Line (15) follows from the induction hypothesis. Line (16) follows from (12).

Line (17) follows because, according to the recursive formula (9), $\boldsymbol{v}_k$ lies in the affine space $\boldsymbol{x}_0 + \text{span}\{\nabla f(\boldsymbol{x}_0), \cdots, \nabla f(\boldsymbol{x}_{k-1})\}$. Similarly, $\boldsymbol{x}_k$ lies in this space. Therefore, $\boldsymbol{v}_k - \boldsymbol{x}_k$ lies the $k$th Krylov subspace $\text{span}\{\nabla f(\boldsymbol{x}_0), \cdots, \nabla f(\boldsymbol{x}_{k-1})\}$. It is well known (see, e.g., [5]) that $\nabla f(\boldsymbol{x}_k)$ is orthogonal to every vector in this space.

Line (18) is a standard inequality for $L$-smooth convex functions; see, e.g., p. 57 of [15]. Finally (19) follows because $\boldsymbol{x}_k - \nabla f(\boldsymbol{x}_k)/L$ lies in the $k+1$-dimensional affine space $\boldsymbol{x}_0 + \text{span}\{\nabla f(\boldsymbol{x}_0), \ldots, \nabla f(\boldsymbol{x}_k)\}$ while $\boldsymbol{x}_{k+1}$ minimizes $f$ over the same affine space, another well known property of CG. □

We note that this proof yields another proof of Daniel's well known result that conjugate gradient converges at a linear rate with a factor $1 - \text{const}\sqrt{\ell/L}$ per iteration, although the constant is better in Daniel's result compared to the analysis in this section.

A limitation of this proof is that it requires $\boldsymbol{v}_0 = \boldsymbol{x}_0$. In the case that the CG iterations follow a sequence of AG iterations, this equality will not hold. We return to this point in Section 5.

## 4 Switching to AG in the case of lack of progress

The C+AG method, if it detects that sufficient progress is not made in the $k$th iteration (i.e., the inequality $f(\boldsymbol{x}_k) \le \phi_k^*$ fails to hold), attempts two fall-back solutions.

The first fall-back is to retry the step using a steepest descent direction instead of the conjugate gradient direction. This is known as "restarting" in the literature on nonlinear conjugate gradient. Restarting in this manner does not play a role in the theoretical analysis of the algorithm, but we found that in practice it improves performance (versus immediately proceeding to the second fall-back of the AG method).

If the inequality $f(\boldsymbol{x}_k) \le \phi_k^*$ fails to hold also for the restart described in the previous paragraph, then the second fall-back is to switch to a sequence of AG steps. We found that AG steps work best if they occur in large blocks (as opposed to individual steps interspersed with CG steps). Furthermore, there are some extra function-gradient evaluations associated with the transition from CG to AG and vice versa. Therefore, once the method switches to an AG step, say at iteration $k$, it continues with AG until an iteration $k'$ is reached such that $\|\nabla f(\boldsymbol{x}_{k'})\| \le \|\nabla f(\boldsymbol{x}_k)\|/4$.

Overall, the method requires two function-gradient evaluations for a CG step, one to obtain the gradient at $\boldsymbol{x}_k$ and the second to evaluate $f(\tilde{\boldsymbol{x}})$. Similarly, the first fall-back requires two function-gradient evaluations. Finally, the accelerated gradient step requires another gradient evaluation. Therefore, the maximum number of function-gradient evaluations on a single iteration before progress is made is five. Therefore, Theorem 1 applies to our algorithm except for a constant factor of 5. In practice, on almost every step, either the CG step succeeds or AG is used, meaning the actual number of function-gradient evaluations per iteration lies between 1 and 2.

## 5 Restarting CG after AG

As mentioned in the previous section, if the sufficient-progress test fails for a CG iteration, then the method can fall back to AG iterations. After a sufficient number of AG iterations to reduce $\|\nabla f(\boldsymbol{x}_k)\|$, the method restarts CG, say at iteration $m$.

An issue with restarting CG is that the sufficient-progress test may not hold for the CG iterations even if the function is purely quadratic on the remaining iterations. This is because the proof of sufficient progress in Theorem 2 assumed that $\boldsymbol{v}_0 = \boldsymbol{x}_0$. However, we would not expect the equality $\boldsymbol{x}_m = \boldsymbol{v}_m$ to hold, where $m$ is the first iteration of restarted CG, and therefore Theorem 2 does not apply.

We address this issue with a more complicated sufficient-progress test for restarted CG. Let $\boldsymbol{z} := \boldsymbol{v}_m - \boldsymbol{x}_m$, assumed to be nonzero. Let us assume that starting from iteration $m$, the function is identically quadratic, say $f(\boldsymbol{x}) = \boldsymbol{x}^T A\boldsymbol{x}/2 - \boldsymbol{b}^T\boldsymbol{x}$. Recall (see, e.g., [5, Chap. 10]) that $\boldsymbol{x}_{m+i} = \operatorname{argmin}\{f(\boldsymbol{x}) : \boldsymbol{x}_m + K_i\}$, where $K_i = \operatorname{span}\{\nabla f(\boldsymbol{x}_m), \ldots, \nabla f(\boldsymbol{x}_{m+i-1})\}$, an affine set. Define $\hat{K}_i := \operatorname{span}\{\nabla f(\boldsymbol{x}_m), \ldots, \nabla f(\boldsymbol{x}_{m+i-1}), \boldsymbol{z}\}$. Let $\bar{\boldsymbol{x}}_{m+i} := \operatorname{argmin}\{f(\boldsymbol{x}) : \boldsymbol{x} \in \boldsymbol{x}_m + \hat{K}_i\}$. We explain below how to efficiently compute $\bar{\boldsymbol{x}}_{m+i}$. The sufficient-progress test we propose is the inequality $f(\bar{\boldsymbol{x}}_{m+i}) \le \phi_{m+i}^*$. Note that $\bar{\boldsymbol{x}}_{m+i}$ appears explicitly on the LHS of this inequality, and the sequence of $\bar{\boldsymbol{x}}_m, \bar{\boldsymbol{x}}_{m+1}, \ldots$ defined in this paragraph is implicit in the RHS $\phi_{m+i}^*$ according to (10).

We now redo the sequence of inequalities used in the proof of Theorem 2 to account for the new definition of $\bar{\boldsymbol{x}}_{m+i}$. First, the base of the induction is that $f(\bar{\boldsymbol{x}}_m) \le f(\boldsymbol{x}_m) \le \phi_m^*$. The first inequality in this chain follows because $\bar{\boldsymbol{x}}_m$ is the minimizer of $f$ over a 1-dimensional affine space that contains $\boldsymbol{x}_m$. The second inequality follows because $f(\boldsymbol{x}_k) \le \phi_k^*$ on every iteration of AG, and we defined $m$ to be the last iterate computed by AG before restarting CG. As for the induction, consider an $i \geq 0$. We start again with (10), now assuming the new formula for $\bar{\boldsymbol{x}}_{m+i}$.

$$
\begin{aligned}
\phi^*_{m+i+1} &= (1-\theta_{m+i})\phi^*_{m+i} + \theta_{m+i} f(\bar{\boldsymbol{x}}_{m+i}) - \frac{\theta^2_{m+i}}{2\gamma_{m+i+1}}\|\nabla f(\bar{\boldsymbol{x}}_{m+i})\|^2 \\
&\quad + \frac{\theta_{m+i}(1-\theta_{m+i})\gamma_{m+i}}{\gamma_{m+i+1}}(\ell\|\bar{\boldsymbol{x}}_{m+i} - \boldsymbol{v}_{m+i}\|^2/2 + \nabla f(\bar{\boldsymbol{x}}_{m+i})^T(\boldsymbol{v}_{m+i} - \bar{\boldsymbol{x}}_{m+i})) \\
&\geq f(\bar{\boldsymbol{x}}_{m+i}) - \frac{\theta^2_{m+i}}{2\gamma_{m+i+1}}\|\nabla f(\bar{\boldsymbol{x}}_{m+i})\|^2 \\
&\quad + \frac{\theta_{m+i}(1-\theta_{m+i})\gamma_{m+i}}{\gamma_{m+i+1}}\nabla f(\bar{\boldsymbol{x}}_{m+i})^T(\boldsymbol{v}_{m+i} - \bar{\boldsymbol{x}}_{m+i}) &(20)\\
&= f(\bar{\boldsymbol{x}}_{m+i}) - \frac{1}{2L}\|\nabla f(\boldsymbol{x}_{m+i})\|^2 \\
&\quad + \frac{\theta_{m+i}(1-\theta_{m+i})\gamma_{m+i}}{\gamma_{m+i+1}}\nabla f(\bar{\boldsymbol{x}}_{m+i})^T(\boldsymbol{v}_{m+i} - \bar{\boldsymbol{x}}_{m+i}). &(21)\\
&= f(\bar{\boldsymbol{x}}_{m+i}) - \frac{1}{2L}\|\nabla f(\bar{\boldsymbol{x}}_{m+i})\|^2 &(22)\\
&\geq f(\bar{\boldsymbol{x}}_{m+i} - \nabla f(\boldsymbol{x}_{m+i})/L) &(23)\\
&\geq f(\bar{\boldsymbol{x}}_{m+i+1}). &(24)
\end{aligned}
$$

Here, line (20) follows from the induction hypothesis. Line (21) follows from (12).

Line (22) follows because $\boldsymbol{v}_{m+i} - \bar{\boldsymbol{x}}_{m+i}$ lies in $\hat{K}_i$. Since $\bar{\boldsymbol{x}}_{m+i}$ is the optimizer of $f$ over $\boldsymbol{x}_m + \hat{K}_i$, it follows that the gradient $\nabla f(\bar{\boldsymbol{x}}_{m+i})$ is orthogonal to any vector in $\hat{K}_i$.

Line (23) again is a standard inequality for $L$-smooth functions. Finally (24) follows because $\boldsymbol{x}_{m+i} - \nabla f(\boldsymbol{x}_{m+i})/L$ lies in $\boldsymbol{x}_m + \hat{K}_{i+1}$, while $\bar{\boldsymbol{x}}_{m+i+1}$ minimizes $f$ over the same affine space.

Having established that the progress measure holds on every iteration for restarted CG, we now turn to the question of computation of $\bar{\boldsymbol{x}}_{m+i}$. We first explain this computation for the quadratic case, and below we extend it to the general case.

Standard theory of LCG (see, e.g., [5, Chap. 10]) states that when started on iteration $m$, a sequence of search directions $\boldsymbol{p}_{m+1}, \boldsymbol{p}_{m+2}, \ldots$ is generated that are mutually conjugate, i.e., $\boldsymbol{p}^T_{m+i} A \boldsymbol{p}_{m+i'} = 0$ provided $i \neq i'$. Furthermore, conjugacy implies that $f$ can be optimized separately with respect to each $\boldsymbol{p}_{m+i}$ to yield an iterate $\boldsymbol{x}_{m+s}$ that minimizes $f$ over the entire affine space $\boldsymbol{x}_m + \operatorname{span}\{\boldsymbol{p}_{m+1}, \ldots, \boldsymbol{p}_{m+s}\} = \boldsymbol{x}_m + K_s$. Therefore, in order to optimize $f$ efficiently over $\boldsymbol{x}_m + \hat{K}_s = \boldsymbol{x}_m + \operatorname{span}\{\boldsymbol{p}_{m+1}, \ldots, \boldsymbol{p}_{m+s}, \boldsymbol{z}\}$, it suffices to construct a $\tilde{\boldsymbol{z}}_s$ such that

1. $\operatorname{span}\{\boldsymbol{p}_{m+1}, \ldots, \boldsymbol{p}_{m+s}, \boldsymbol{z}\} = \operatorname{span}\{\boldsymbol{p}_{m+1}, \ldots, \boldsymbol{p}_{m+s}, \tilde{\boldsymbol{z}}_s\}$, and

2. $\tilde{\boldsymbol{z}}_s$ is conjugate to $\boldsymbol{p}_{m+1}, \ldots, \boldsymbol{p}_{m+s}$.

In this case, to optimize $f$ over $\boldsymbol{x}_m + \hat{K}_s$ is straightforward: one takes a solution of the form $\boldsymbol{x}_{m+s} + \tilde{\alpha}_s \tilde{\boldsymbol{z}}_s$, where $\tilde{\alpha}_s$ is the minimizer of the univariate quadratic function $\alpha \mapsto f(\boldsymbol{x}_{m+s} + \alpha \tilde{\boldsymbol{z}}_s)$. The solution in closed form is

$$\tilde{\alpha}_s := -\nabla f(\boldsymbol{x}_{m+s})^T \tilde{\boldsymbol{z}}_s / (\tilde{\boldsymbol{z}}_s^T A \tilde{\boldsymbol{z}}_s). \tag{25}$$

The vector $\tilde{\boldsymbol{z}}_s$ can be computed via a recurrence relationship as follows. The base of the recurrence is $\tilde{\boldsymbol{z}}_0 := \boldsymbol{z}$, which satisfies the two conditions for $\tilde{\boldsymbol{z}}_s$ stated in the last paragraph. Now, suppose $\tilde{\boldsymbol{z}}_s$ satisfies the two conditions. We seek a scalar $\delta_{s+1}$ such that the formula $\tilde{\boldsymbol{z}}_{s+1} := \tilde{\boldsymbol{z}}_s - \delta_{s+1}\boldsymbol{p}_{m+s+1}$ satisfies the two conditions. It is clear that regardless of how $\delta_{s+1}$ is defined, condition 1 is satisfied. Also, regardless of how $\delta_{s+1}$ is chosen, conjugacy of $\tilde{\boldsymbol{z}}_{s+1}$ with respect to $\boldsymbol{p}_{m+1},\ldots,\boldsymbol{p}_{m+s}$ is assured since both $\tilde{\boldsymbol{z}}_s$ and $\boldsymbol{p}_{m+s+1}$ are conjugate to these $s$ vectors. Thus, to conclude condition 2, let us choose $\delta_{s+1}$ to ensure conjugacy with $\boldsymbol{p}_{m+s+1}$, i.e., solve the equation $(\tilde{\boldsymbol{z}}_s - \delta_{s+1}\boldsymbol{p}_{m+s+1})^T A\boldsymbol{p}_{m+s+1} = 0$, which has as its closed form solution:

$$\delta_{s+1} := \tilde{\boldsymbol{z}}_s^T A\boldsymbol{p}_{m+s+1}/(\boldsymbol{p}_{m+s+1}^T A\boldsymbol{p}_{m+s+1}) \tag{26}$$

Notice that there is also a recurrent formula for $\tilde{\boldsymbol{z}}_s^T A\tilde{\boldsymbol{z}}_s$ that appears in (25). In particular, one checks that

$$\tilde{\boldsymbol{z}}_{s+1}^T A\tilde{\boldsymbol{z}}_{s+1} = \tilde{\boldsymbol{z}}_s^T A\tilde{\boldsymbol{z}}_s - \delta_{s+1}^2\tilde{\boldsymbol{z}}_s^T A\boldsymbol{p}_{m+s+1} + \delta_{s+1}^2\boldsymbol{p}_{m+s+1}^T A\boldsymbol{p}_{m+s+1}. \tag{27}$$

The second and third terms on the right-hand side can be simplified by substituting the formula for $\delta_{s+1}$.

This concludes the discussion of how to compute $\bar{\boldsymbol{x}}_{m+i}$ in the case that $f$ is quadratic. In the general case, we do not have a matrix $A$ but only $f$ and $\nabla f$. Therefore, we propose a method to evaluate the quantities in the preceding paragraphs in the general case that reduces to the given formulas for the quadratic case. In the general case, we again initialize $\tilde{\boldsymbol{z}}_0 = \boldsymbol{z} = \boldsymbol{v}_m - \boldsymbol{x}_m$. Nonlinear CG has access to $\boldsymbol{p}_{m+i}$, and we are already computed a generalization of $A\boldsymbol{p}_{m+i}$ and $\boldsymbol{p}_{m+i}^T A\boldsymbol{p}_{m+i}$ in (3)–(5). Therefore, we can compute both the numerator and denominator of the right-hand side of (26).

As for $\tilde{\boldsymbol{z}}_s^T A\boldsymbol{z}_s$, on the first iteration (i.e., $s = 0$, that is, iteration $m$), we obtain $\boldsymbol{z}^T A\boldsymbol{z}$ by the formula $(\nabla f(\boldsymbol{v}_m) - \nabla f(\boldsymbol{x}_m))^T\boldsymbol{z}$. This requires an extra gradient evaluation. After this computation, we can update $\tilde{\boldsymbol{z}}_s^T A\tilde{\boldsymbol{z}}_s$ using the recurrent formula (27).

Thus, maintaining the quantity $\tilde{\boldsymbol{z}}_s$ does not require any extra function-gradient evaluations after the first CG iteration. However, the algorithm needs to evaluate $f(\bar{\boldsymbol{x}}_k)$ in order to check that $f(\bar{\boldsymbol{x}}_k) \le \phi_k^*$. It also needs $\nabla f(\bar{\boldsymbol{x}}_k)$ in order to evaluate (8)–(10). Thus, the method described in this section requires three function-gradient evaluations for a typical CG iteration rather than two that were described earlier. We return to the matter of this 50% overhead in Section 8.

## 6 Toward $n$-step quadratic convergence

In the previous sections, we showed the proposed C+AG method has the same complexity bound up to a constant factor as accelerated gradient. In this section we suggest that it has an additional complexity bound known to apply to nonlinear conjugate gradient, namely, $n$-step quadratic convergence. This property is mainly of theoretical interest since $n$-step quadratic convergence is not usually observed in practice (see, e.g., [16, p. 124]) for larger problems. Therefore, we will provide only an outline of a possible proof and leave the proof, which is likely to be complicated, to future research.

We propose to follow the proof given by McCormick and Ritter [12] rather than [3]. They argue that classical NCG is $n$-step quadratically convergent provided that the line-search is sufficiently accurate and provided that the objective function is strongly convex in the neighborhood of the objective function and that its second derivative is Lipschitz continuous. Finally, they assume that nonlinear CG is restarted every $n$ iterations.

In order to apply this result to C+AG, we first need to argue that the sufficient-progress test will not prevent CG iterations. We showed in the last section that for quadratic functions, the sufficient-progress test always holds. A strongly convex function with a Lipschitz second derivative is approximated arbitrarily closely by a quadratic function in a sufficiently small neighborhood of the root. In particular, if we assume that $L$ is strictly larger than the largest eigenvalue of the Hessian in a neighborhood of the root, then inequality (23) is strictly satisfied, and small deviations from a quadratic function are dominated by the residual in (23).

Some other issues with [12] is that their version of NCG implements a step-limiter operation that is not present in our algorithm. It is not clear whether the step-limiter is a requirement for $n$-step quadratic convergence or merely a device to simplify their analysis. Furthermore, [12] assume the Polak-Ribière choice of $\beta_k$ [16] rather than the Hager-Zhang step that we used. Note that [12] predates [7] by several decades. We conjecture that the analysis of [12] extends to [7].

The line-search accuracy required by [12] is as follows. On each iteration $k$, $|\alpha_{k+1} - \alpha^*_{k+1}| \le O(\|\boldsymbol{p}_{k+1}\|)$, where $\alpha_{k+1}$ is chosen by C+AG in (3)–(5) and $\alpha^*_{k+1} = \operatorname{argmin}\{f(\boldsymbol{x}_k + \alpha\boldsymbol{p}_{k+1}) : \alpha \in \mathbb{R}\}$. We sketch an argument to support this as follows.

Let $\psi(\alpha) := f(\boldsymbol{x}_k + \alpha\boldsymbol{p}_{k+1})$. It follows from [12] that $\|\boldsymbol{p}_{k+1}\| = \Theta(\|\boldsymbol{x}_k - \boldsymbol{x}^*\|)$. In other words, there is both an upper and lower bound on

$$\frac{\|\boldsymbol{p}_{k+1}\|}{\|\boldsymbol{x}_k - \boldsymbol{x}^*\|}$$

independent of $k$. It also follows from their results that $\psi'(0) < 0$. Write $\psi''(\alpha) = s + \epsilon(\alpha)$, where $s > 0$ is the estimate of $\psi''$ obtained from (3)–(5), that is $s = L(\psi'(1/L) - \psi'(0))$. It follows by strong convexity and the equation $\|\boldsymbol{p}_{k+1}\| = \Theta(\|\boldsymbol{x}_k - \boldsymbol{x}^*\|)$ that $\alpha_{k+1} \le O(1)$ and $\alpha^*_{k+1} \le O(1)$. Again by these same assumptions, we can also derive that $s = \Theta(\|\boldsymbol{p}_{k+1}\|^2)$. By the Lipschitz assumption, $\epsilon(\alpha) \le O(\|\boldsymbol{p}_{k+1}\|^3)$ for $\alpha \le O(1)$.

It also follows from the assumption of strict convexity and proximity to the root $|\psi'(0)| \le O(\|\boldsymbol{p}_{k+1}\|^2)$. The true minimizer $\alpha^*_{k+1}$ is the root of $\psi'$ and therefore satisfies the equation

$$\psi'(0) + \int_0^{\alpha^*_{k+1}} (s + \epsilon(\alpha))\, d\alpha = 0,$$

i.e.,

$$\psi'(0) + s\alpha^*_{k+1} + \int_0^{\alpha^*_{k+1}} \epsilon(\alpha)\, d\alpha = 0.$$

The computed minimizer is $\alpha_{k+1} = -\psi'(0)/s$. The integral in the previous line is bounded by $O(\|\boldsymbol{p}_{k+1}\|^3)$, whereas $-\psi'(0)$ is $O(\|\boldsymbol{p}_{k+1}\|^2)$, and $s = \Theta(\|\boldsymbol{p}_{k+1}\|^2)$. Therefore, the intermediate value theorem applied to the above univariate equation for $\alpha^*_{k+1}$ tells us that

$\alpha^*_{k+1} = -\psi'(0)/s + O(\|\boldsymbol{p}_{k+1}\|) = \alpha_k + O(\|\boldsymbol{p}_{k+1}\|)$. This concludes the explanation of accuracy of the line-search.

## 7 Pseudocode for C+AG

In this section we present the complete pseudocode for the C+AG algorithm. See the concluding remarks for a downloadable implementation in Matlab. The input arguments are $f$ and $\nabla f$, the objective function and gradient, $\boldsymbol{x}_0$, the initial guess, `gtol`, the termination criterion, and $L$, $\ell$, the smoothness modulus and the modulus of strong convexity respectively. Note that $\ell = 0$ is a valid input.

**Algorithm 1** ComputeThetaGamma

```
Input: $L, \ell, \gamma_k$
Output: $\theta_k, \gamma_{k+1}$
  Solve $L\theta_k^2 + (\gamma_k - \ell)\theta_k - \gamma_k = 0$ via the quadratic formula for the positive root $\theta_k$.
  Let $\gamma_{k+1} := (1-\theta_k)\gamma_k + \theta_k\ell$
```

**Algorithm 2** ComputeVPhiStar

```
Input: $\theta_k, \gamma_k, \gamma_{k+1}, \ell, \boldsymbol{v}_k, \phi_k^*, \bar{\boldsymbol{x}}_k, f(\bar{\boldsymbol{x}}_k), \nabla f(\bar{\boldsymbol{x}}_k)$
Output: $\boldsymbol{v}_{k+1}, \phi^*_{k+1}$
  Let $\boldsymbol{v}_{k+1} := \frac{1}{\gamma_{k+1}}[(1-\theta_k)\gamma_k\boldsymbol{v}_k + \theta_k\ell\bar{\boldsymbol{x}}_k - \theta_k\nabla f(\bar{\boldsymbol{x}}_k)]$.
  Let $\phi^*_{k+1} := (1-\theta_k)\phi_k^* + \theta_k f(\bar{\boldsymbol{x}}_k) - \frac{\theta_k^2}{2\gamma_{k+1}}\|\nabla f(\bar{\boldsymbol{x}}_k)\|^2$
      $+ \frac{\theta_k(1-\theta_k)\gamma_k}{\gamma_{k+1}}(\ell\|\bar{\boldsymbol{x}}_k - \boldsymbol{v}_k\|^2/2 + \nabla f(\bar{\boldsymbol{x}}_k)^T(\boldsymbol{v}_k - \bar{\boldsymbol{x}}_k))$.
```

The mathematical structure of the algorithm has already been described in previous sections, and in this section we will describe a few computational details. The flag `onlyAG` indicates that the algorithm is currently executing a block of AG statements. As mentioned in Section 4, once AG iterations begin, they continue uninterrupted until $\|\nabla f(\bar{\boldsymbol{x}}_k)\| \le \|\nabla f(\boldsymbol{x}_{k_{AG}})\|/4$, where $k_{AG}$ denotes the first iteration of the block of AG iterations. The flag `zflag` indicates that $\tilde{\boldsymbol{z}}$ has been selected. For the first block of consecutive CG statements, this flag is false, meaning that l. 21 defines $\bar{\boldsymbol{x}}_k := \boldsymbol{x}_k$ as in Section 3 When CG is re-entered following a block of AG statements, this flag is true, meaning that ll. 15–18 define $\bar{\boldsymbol{x}}_k$ is according to Section 5 involving the computation of $\tilde{\boldsymbol{z}}$.

The variable $i_{cg}$ counts the number of consecutive CG iterations without a restart. A restart means that $\boldsymbol{p}_{k+1}$ is defined to be $-\boldsymbol{g}_k$, i.e., a steepest descent step is taken. The code forces a restart after $10n + 1$ iterations. Such a test is typical in NCG algorithms. The theory for $n$-step quadratic convergence requires a restart every $n + 1$ iterations, but well known codes, e.g., CG-Descent [7] often use a longer interval.

**Algorithm 3** C+AG (Part I)

```
Input: $f(\cdot), \nabla f(\cdot), \boldsymbol{x}_0 \in \mathbb{R}^n, \ell, L, \texttt{gtol}$
Output: $\boldsymbol{x}^*$ such that $\|\nabla f(\boldsymbol{x}^*)\| \le \texttt{gtol}$
 1: $f_0 := f(\boldsymbol{x}_0)$; $\boldsymbol{g}_0 := \nabla f(\boldsymbol{x}_0)$; $\phi_0^* := f_0$; $\boldsymbol{v}_0 := \boldsymbol{x}_0$; $\gamma_0 := L$; $\texttt{onlyAG} :=$ false;
 2: $i_{cg} := 0$; $\texttt{zflag} :=$ false; $\bar{\boldsymbol{x}}_0 := \boldsymbol{x}_0$; $\bar{f}_0 := f_0$; $\bar{\boldsymbol{g}}_0 := \boldsymbol{g}_0$; $\boldsymbol{p}_1 := -\boldsymbol{g}_0$
 3: for $k = 0, 1, \ldots$ do
 4:     if $i_{cg} \ge 10n + 1$ then $\boldsymbol{p}_{k+1} := -\boldsymbol{g}_k$; $i_{cg} := 0$ end if
 5:     $\theta_k, \gamma_{k+1} := \texttt{ComputeThetaGamma}(L, \ell, \gamma_k)$
 6:     if not onlyAG then
 7:         for $\texttt{cgAttempt} = 1 : 2$ do
 8:             if $\texttt{cgAttempt} = 2$ then $\boldsymbol{p}_{k+1} := -\boldsymbol{g}_k$; $i_{cg} := 0$ end if
 9:             $\tilde{\boldsymbol{x}} := \boldsymbol{x}_k + \boldsymbol{p}_{k+1}/L$; $\tilde{f} := f(\tilde{\boldsymbol{x}})$; $\tilde{\boldsymbol{g}} := \nabla f(\tilde{\boldsymbol{x}})$
10:             if $\|\tilde{\boldsymbol{g}}\| \le \texttt{gtol}$ then return $\tilde{\boldsymbol{x}}$ end if
11:             $\texttt{Ap} := L(\tilde{\boldsymbol{g}} - \boldsymbol{g}_k)$; $\texttt{pAp} := \boldsymbol{p}_{k+1}^T\texttt{Ap}$; $\alpha_{k+1} := -\boldsymbol{g}^T\boldsymbol{p}_{k+1}/\texttt{pAp}$
12:             $\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \alpha_{k+1}\boldsymbol{p}_{k+1}$; $f_{k+1} := f(\boldsymbol{x}_{k+1})$; $\boldsymbol{g}_{k+1} := \nabla f(\boldsymbol{x}_{k+1})$
13:             if $\|\boldsymbol{g}_{k+1}\| \le \texttt{gtol}$ then return $\boldsymbol{x}_{k+1}$ end if
14:             if zflag then
15:                 $\texttt{zAp} := \tilde{\boldsymbol{z}}^T\texttt{Ap}$; $\delta := \texttt{zAp}/\texttt{pAp}$; $\tilde{\boldsymbol{z}} := \tilde{\boldsymbol{z}} - \delta\boldsymbol{p}_{k+1}$
16:                 $\texttt{zAz} := \texttt{zAz} - 2\delta \cdot \texttt{zAp} + \delta^2\texttt{pAp}$
17:                 $\tilde{\alpha} := -\nabla f(\boldsymbol{x}_{k+1})^T\tilde{\boldsymbol{z}}/\texttt{zAz}$
18:                 $\bar{\boldsymbol{x}}_{k+1} := \bar{\boldsymbol{x}}_{k+1} + \tilde{\alpha}\tilde{\boldsymbol{z}}$; $\bar{f}_{k+1} := f(\bar{\boldsymbol{x}}_{k+1})$; $\bar{\boldsymbol{g}}_{k+1} := \nabla f(\bar{\boldsymbol{x}}_{k+1})$;
19:                 if $\|\bar{\boldsymbol{g}}_{k+1}\| \le \texttt{gtol}$ then return $\bar{\boldsymbol{x}}_{k+1}$ end if
20:             else
21:                 $\bar{\boldsymbol{x}}_{k+1} := \boldsymbol{x}_{k+1}$; $\bar{f}_{k+1} := f_{k+1}$; $\bar{\boldsymbol{g}}_{k+1} := \boldsymbol{g}_{k+1}$
22:             end if
23:             $\boldsymbol{v}_{k+1}, \phi_{k+1}^* := \texttt{ComputeVPhiStar}(\theta_k, \gamma_k, \gamma_{k+1}, \ell, \boldsymbol{v}_k, \phi_k^*, \bar{\boldsymbol{x}}_k, \bar{f}_k, \bar{\boldsymbol{g}}_k)$
24:             if $f_{k+1} \le \phi_{k+1}^*$ or $\bar{f}_{k+1} \le \phi_{k+1}^*$ then
25:                 $\hat{\boldsymbol{y}} := \boldsymbol{g}_{k+1} - \boldsymbol{g}_k$
26:                 $\beta^1 := \left(\hat{\boldsymbol{y}} - \boldsymbol{p}_{k+1} \cdot \frac{2\|\hat{\boldsymbol{y}}\|^2}{\hat{\boldsymbol{y}}^T\boldsymbol{p}_{k+1}}\right)^T \frac{\boldsymbol{g}_{k+1}}{\hat{\boldsymbol{y}}^T\boldsymbol{p}_{k+1}}$
27:                 $\beta^2 := \frac{-1}{\|\boldsymbol{p}_{k+1}\| \cdot \min(.01\|\boldsymbol{g}_0\|, \|\boldsymbol{g}_{k+1}\|)}$.
28:                 $\beta_{k+1} := \max(\beta^1, \beta^2)$
29:                 $\boldsymbol{p}_{k+2} := -\boldsymbol{g}_{k+1} + \beta_{k+1}\boldsymbol{p}_{k+1}$
30:                 $i_{cg} := i_{cg} + 1$
31:                 if $\beta_{k+1} = 0$ then $i_{cg} := 0$ end if
32:                 goto IterationEnd
33:             end if
34:         end for
35:     end if
```

**Algorithm 4** C+AG Part II

```
36:     if not onlyAG then
37:         onlyAG := true
38:         $k_{AG} := k$
39:     end if
40:     $\bar{\boldsymbol{x}}_k := \frac{\theta_k \gamma_k \boldsymbol{v}_k + \gamma_{k+1} \boldsymbol{x}_k}{\gamma_k + \theta_k \ell}$; $\bar{f}_k := f(\bar{\boldsymbol{x}}_k)$; $\bar{\boldsymbol{g}}_k := \nabla f(\bar{\boldsymbol{x}}_k)$
41:     if $\|\bar{\boldsymbol{g}}_k\| \le$ gtol then return $\bar{\boldsymbol{x}}_k$ end if
42:     $\boldsymbol{x}_{k+1} := \bar{\boldsymbol{x}}_k - \bar{\boldsymbol{g}}_k / L$
43:     $\boldsymbol{v}_{k+1}, \phi^*_{k+1} :=$ ComputeVPhiStar($\theta_k, \gamma_k, \gamma_{k+1}, \ell, \boldsymbol{v}_k, \phi^*_k, \bar{\boldsymbol{x}}_k, \bar{f}_k, \bar{\boldsymbol{g}}_k$)
44:     if $\|\bar{\boldsymbol{g}}_k\| \le \|\bar{\boldsymbol{g}}_{k_{AG}}\|/4$ then
45:         $f_{k+1} := f(\boldsymbol{x}_{k+1})$; $\boldsymbol{g}_{k+1} := \nabla f(\boldsymbol{x}_{k+1})$
46:         $\bar{\boldsymbol{x}}_{k+1} := \boldsymbol{x}_{k+1}$; $\bar{f}_{k+1} := f_{k+1}$; $\bar{\boldsymbol{g}}_{k+1} := \boldsymbol{g}_{k+1}$
47:         $\tilde{\boldsymbol{z}} := \boldsymbol{v}_{k+1} - \boldsymbol{x}_{k+1}$
48:         zflag := true
49:         $f^v := f(\boldsymbol{v}_{k+1})$; $\boldsymbol{g}^v := \nabla g(\boldsymbol{v}_{k+1})$
50:         zAz := $\tilde{\boldsymbol{z}}^T(\boldsymbol{g}^v - \boldsymbol{g}_{k+1})$
51:         $\boldsymbol{p}_{k+2} := -\boldsymbol{g}_{k+1}$
52:         onlyAG := false
53:     end if
54:     IterationEnd:
55: end for
```

The for-loop on l. 7 tries first a CG step and second a steepest descent step in order to make progress. Line 11 computes $\alpha_{k+1}$ as discussed in Section 2. The test for sufficient progress in a CG step appears in l. 24. If sufficient progress is attained, then the next CG search direction $\boldsymbol{p}_{k+2}$ is computed.

Line 36 can be reached if both attempts at CG fail to make sufficient progress. It is also reached if the onlyAG flag is set to 'true', meaning that the algorithm is currently inside a block of consecutive AG iterations. The statements beginning with 40 are the formulation of the AG method taken from [15]. The statements beginning on l. 45 set the variables to return to CG iterations including initilization of $\tilde{\boldsymbol{z}}$ and zAz, which stands for $\tilde{\boldsymbol{z}}^T A \tilde{\boldsymbol{z}}$ in the quadratic case.

## 8 Computational experiments

In this section we report on computational experiments with C+AG. We compared it to two other codes: AG coded by us based on (2.2.8) of [15] and CG-Descent by Hager and Zhang. We used C version 6.8 of CG-Descent with the Matlab front-end. We used it in memoryless mode, i.e., NCG rather than L-BFGS.

As for our own code C+AG, we discovered via experimentation that the more complex calculation of $\bar{\boldsymbol{x}}_k$ described in Section 5 diminished the performance of the code. In particular, as mentioned in Section 5, the additional overhead of 50% per iteration to evaluate $f(\bar{\boldsymbol{x}}_k)$ and $\nabla f(\bar{\boldsymbol{x}}_k)$ on l. 18 slowed the code down without improving the iteration count. We found that using the progress measure based on taking $\bar{\boldsymbol{x}}_k := \boldsymbol{x}_k$ as in Section 3 worked well even after restarting CG despite the issue that $\boldsymbol{v}_m \neq \boldsymbol{x}_m$, meaning that the proof of Theorem 2 is no longer valid for restarted CG. Therefore, our tests in this section have modified Algorithm C+AG by deleting statements 47 to 50, meaning that $\tilde{\boldsymbol{z}}$ is never defined and `zflag` is never changed to 'true'.

Removal of these statements does not affect the following three properties of the algorithm: optimal convergence rate for smooth convex functions, optimal convergence rate for smooth strongly convex functions, equivalent to LCG for quadratic functions. However, removal of these statements may prevent the method from returning from AG to CG steps for some problems, meaning that our conjectured $n$-step quadratic convergence result may no longer hold. Since we did not observe $n$-step quadratic convergence in practice, we are not able to investigate this issue computationally.

To the best of our knowledge, there is no standard benchmark set of smooth convex functions with a known $L, \ell$, so instead we constructed our own test set of eight problem instances based on formulations that have occurred in the recent literature in data science.

In all problems, the convergence criterion is $\|\nabla f(\boldsymbol{x}_k)\| \leq 10^{-8}$. In the case of CG-Descent, the convergence criterion is based on the $\infty$-norm rather than 2-norm. Via some experiments not reported here, we found that stopping when $\|\nabla f(\boldsymbol{x}_k)\|_\infty \leq 3 \cdot 10^{-8}/\sqrt{n}$ approximately reproduced the criterion $\|\nabla f(\boldsymbol{x}_k)\| \leq 10^{-8}$ used for the other methods, so this was the termination criterion for CG-Descent.

The first four problems are based on BPDN (basis pursuit denoising), that is, the unconstrained convex optimization problem:

$$\min \frac{1}{2}\|A\boldsymbol{x} - \boldsymbol{b}\|^2 + \lambda\|\boldsymbol{x}\|_1$$

in which $\lambda > 0$ and $A \in \mathbb{R}^{m \times n}$ has fewer rows than columns, so that the problem is neither strongly convex nor smooth. However, the following approximation (called APBDN) is both smooth and strongly convex on any bounded domain:

$$\min \frac{1}{2}\|A\boldsymbol{x} - \boldsymbol{b}\|^2 + \lambda \sum_{i=1}^{n} \sqrt{x_i^2 + \delta}$$

where $\delta > 0$ is a fixed scalar. It is easy to see that as $\delta \to 0$, the original problem is recovered. As $\delta \to 0$, $\ell \to 0$ and $L \to \infty$, where $\ell, L$ are the moduli of strong and smooth convexity respectively.

In our tests of ABPDN we took $A$ to be a subset of $\sqrt{n}$ rows of the discrete-cosine transform matrix of size $n \times n$, where $n$ is an even power of 2. (This matrix and its transpose, although dense, can be applied in $O(n \ln n)$ operations.) The subset of rows was selected to be those numbered by the first $m = \sqrt{n}$ prime integers in order to get reproducible pseudorandomness in the choices. Similarly, in order to obtain a pseudorandom $\boldsymbol{b}$, we selected $\boldsymbol{b} \in \mathbb{R}^m$ according to the formula $b_i = \sin(i^2)$. The value of $\lambda$ was fixed at $10^{-3}$ in all tests. Finally, we varied $\delta = 10^{-4}$ and $\delta = 5 \cdot 10^{-6}$ and we tried both $n = 65536$ and $n = 262144$ for a total of four test cases.

Table 2: Function-gradient evaluation counts for three algorithms on eight problems. Bold indicates the best for each row.

| Problem | C+AG | AG | CG-Descent |
|---|---:|---:|---:|
| ABPDN($n = 65536, \delta = 10^{-4}$) | **56,287** | 518,020 | 83,808 |
| ABPDN($n = 65536, \delta = 5 \cdot 10^{-6}$) | **124,066** | 660,356 | 158,236 |
| ABPDN($n = 262144, \delta = 10^{-4}$) | **80,084** | 901,419 | 130,951 |
| ABPDN($n = 262144, \delta = 5 \cdot 10^{-6}$) | **484,639** | $> 10^6$ | 694,216 |
| LL($\lambda = 10^{-4}$) | **118** | 106,506 | 132 |
| LL($\lambda = 5 \cdot 10^{-6}$) | **110** | 362,236 | 131 |
| HR($\tau = 250$) | **4,580** | 7,219 | 42,047 |
| HR($\tau = 1000$) | 57,018 | **51,900** | 513,648 |

The second test case is logistic loss (LL), which is as follows:

$$f(\boldsymbol{x}) = L(A\boldsymbol{x}) + \lambda \|\boldsymbol{x}\|^2/2,$$

where $A$ is a given $m \times n$ matrix, and $\lambda > 0$ is a regularization parameter. Here, $L(\boldsymbol{v}) = \sum_{i=1}^m l(v_i)$ where $l(v) = \ln(1 + e^{-v})$. Row $i$ of $A$, $i = 1 : n$, is of the form $\boldsymbol{e}^T/\sqrt{n} + \boldsymbol{z}_i$, where $\boldsymbol{e} \in \mathbb{R}^n$ is the vector of all 1's and $\boldsymbol{z}_i \sim N(\boldsymbol{0}, \sigma^2 I)$ where $\sigma = 0.4$. We seeded Matlab's random number generator with the same seed on each run for reproducibility. This formulation arises from the problem of identifying the the best halfspace that contains noisy data points. We ran this test with $A \in \mathbb{R}^{6000 \times 3000}$ with two values of $\lambda$, namely, $\lambda = 10^{-4}$ and $\lambda = 5 \cdot 10^{-6}$. This function is smooth and strongly convex. The function $\lambda$ controls the strong convexity.

The third test case is Huber regression (HR). Given a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $\boldsymbol{b} \in \mathbb{R}^m$, Huber regression [9] has been proposed as a means to make linear least-squares regression more robust against outliers. In more detail, a cutoff $\tau > 0$ is selected in order to define the function

$$\zeta(t) = \begin{cases} -\tau^2 - 2\tau t, & t \le -\tau, \\ t^2, & t \in [-\tau, \tau], \\ -\tau^2 + 2\tau t, & t \ge \tau. \end{cases}$$

Note that $\zeta(t)$ is differentiable and convex and behaves like $t^2$ for small $|t|$ and like $O(|t|)$ for large $|t|$. Then, $f(\boldsymbol{x}) = \sum_{i=1}^m \zeta(A(i,:)\boldsymbol{x} - b_i)$. This function is smooth but not strongly convex, i.e., $\ell = 0$. We chose the $(n+1) \times n$ sparse matrix with 1's on the main diagonal and $-1$'s on the first sub diagonal. The vector $\boldsymbol{b}$ was taken to be $1 : n + 1$. We fixed $n = 10000$. We selected two choices for $\tau$, namely, $\tau = 250$ and $\tau = 1000$.

Our results for the three algorithms on the eight problems are shown in Table 2. We set an iteration limit of 500,000 for C+AG and CG-Descent and 1,000,000 for AG (since each AG iteration requires one function-gradient evaluation, whereas an iteration of C+AG or CG-Descent requires approximately two).

The results in Table 2 show that C+AG outperformed both AG and CG-Descent in every case except the last row. We see from this table that for some problems AG significantly outperformed CG-Descent, while for others it was the opposite. But in all cases, C+AG did as about as well as or better than whichever of AG or CG-Descent performed better.

## 9 Conclusions

We have presented an algorithm called C+AG, which is a variant of nonlinear conjugate gradient tailored to smooth, convex objective functions. It has a guaranteed rate of convergence equal to that of accelerated gradient, which is optimal for the class of problems under consideration in the function-gradient evaluation model of computation. The method reduces to linear conjugate gradient in the case that the objective function is quadratic.

The code and all the test cases are implemented in Matlab and available on Github under the project named 'ConjugatePlusAcceleratedGradient'.

The C+AG code outperformed both AG and CG on almost all test cases tried. One interesting future direction is as follows. The method switches between CG and AG steps depending on the progress measure. A more elegant approach would be a formulation that interpolates continuously between the two steps. Indeed, it is possible (see, e.g., [11]) to write down a single parameterized update step in which one parameter value corresponds to AG and another to CG. But we do not know of a systematic way to select this parameter that leads to a complexity bound.

One limitation of the method is the requirement for prior estimates of moduli $L, \ell$ of the objective function. It would be relatively straightforward to implement an adaptive method that estimates $L$ via backtrack line search. There is also literature related to estimating $\ell$. See, e.g., [17], for information on adaptively estimating $L$ and $\ell$.

Another interesting direction is to develop a conjugate gradient iteration suitable for constrained convex problems. Conjugate gradient is known to extend to the special case of minimizing a convex quadratic function with a Euclidean-ball constraint; see e.g. Gould et al. [6] and further remarks on the complexity in Paquette and Vavasis [18].

## References

[1] S. Bubeck, Y. T. Lee, and Mohit Singh. A geometric alternative to Nesterov's accelerated gradient descent. `http://arxiv.org/abs/1506.08187`, 2015.

[2] Yair Carmon, John C. Duchi, Oliver Hinder, and Aaron Sidford. "Convex until proven guilty": dimension-free acceleration of gradient descent on non-convex functions. In *Proceedings of 2017 International Conference on Machine Learning*, 2017.

[3] A. Cohen. Rate of convergence of several conjugate gradient algorithms. *SIAM Journal on Numerical Analysis*, 9(2):248–259, 1972.

[4] James W Daniel. The conjugate gradient method for linear and nonlinear operator equations. *SIAM Journal on Numerical Analysis*, 4(1):10–26, 1967.

[5] G. H. Golub and C. F. Van Loan. *Matrix Computations, 3rd Edition.* Johns Hopkins University Press, Baltimore, 1996.

[6] N. Gould, S. Lucidi, M. Roma, and P. Toint. Solving the trust-region subproblem using the Lanczos method. *SIAM Journal on Optimization*, 9(2):504–525, 1999.

[7] W. Hager and H. Zhang. A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM J. Optimization*, 16:170–192, 2005.

[8] Magnus Rudolph Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[9] P. J. Huber. Robust regression: asymptotics, conjectures, and Monte Carlo. *Annals of Statistics*, 1(5):799–821, 1973.

[10] Sahar Karimi. *On the relationship between conjugate gradient and optimal first-order methods for convex optimization.* PhD thesis, University of Waterloo, 2014.

[11] Sahar Karimi and Stephen Vavasis. A single potential governing convergence of conjugate gradient, accelerated gradient and geometric descent. `https://arxiv.org/abs/1712.09498`, 2017.

[12] G. P. McCormick and K. Ritter. Alternative proofs of the convergence properties of the conjugate-gradient method. *J. Optimization Theory and Applications*, 13(5):497–518, 1974.

[13] A. S. Nemirovsky and D. B. Yudin. *Problem Complexity and Method Efficiency in Optimization.* John Wiley and Sons, Chichester, 1983. Translated by E. R. Dawson from *Slozhnost' Zadach i Effektivnost' Metodov Optimizatsii*, 1979, Glavnaya redaktsiya fiziko-matematicheskoi literatury, Izdatelstva "Nauka".

[14] Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. *Doklady AN SSSR (translated as Soviet Math. Dokl.)*, 269(3):543–547, 1983.

[15] Y. Nesterov. *Introductory Lectures on Convex Optimization.* Kluwer, 2003.

[16] J. Nocedal and S. Wright. *Numerical Optimization, 2nd Edition.* Springer, New York, 2006.

[17] B. O'Donoghue and E. Candès. Adaptive restart for accelerated gradient schemes. *Foundations of Computational Mathematics*, 15(3):715–732, 2015.

[18] C. Paquette and S. Vavasis. Potential-based analyses of first-order methods for constrained and composite optimization. `http://arxiv.org/abs/1903.08497`, 2019.