Math. Prog. Comp. (2018) 10:457–486
https://doi.org/10.1007/s12532-018-0136-7

FULL LENGTH PAPER

Cubic regularization in symmetric rank-1 quasi-Newton methods

Hande Y. Benson[1] · **David F. Shanno**[2]

Received: 16 August 2016 / Accepted: 5 January 2018 / Published online: 12 February 2018

Abstract Quasi-Newton methods based on the symmetric rank-one (SR1) update have been known to be fast and provide better approximations of the true Hessian than popular rank-two approaches, but these properties are guaranteed under certain conditions which frequently do not hold. Additionally, SR1 is plagued by the lack of guarantee of positive definiteness for the Hessian estimate. In this paper, we propose cubic regularization as a remedy to relax the conditions on the proofs of convergence for both speed and accuracy and to provide a positive definite approximation at each step. We show that the n-step convergence property for strictly convex quadratic programs is retained by the proposed approach. Extensive numerical results on unconstrained problems from the CUTEr test set are provided to demonstrate the computational efficiency and robustness of the approach.

Keywords Interior-point methods · Nonlinear programming · Cubic regularization · Newton's method

Mathematics Subject Classification 90C30 · 90C53 · 90–08

1 Introduction

In this paper, we consider the unconstrained nonlinear programming problem (NLP)

$$\min_x f(x), \tag{1}$$

✉ Hande Y. Benson
hvb22@drexel.edu

1 Drexel University, Philadelphia, PA, USA

2 Emeritus, Rutgers University, New Brunswick, NJ, USA

where $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$. We assume that f is smooth and that its Hessian is Lipschitz continuous on at least the set $\{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$.

Many algorithms for solving (1) employ a quadratic model of the problem. Newton's method and quasi-Newton methods use a second-order Taylor series expansion of the objective function, with either an explicit or approximated Hessian matrix. In theory and practice, these algorithms are robust on problems that obey certain assumptions and exhibit super-linear rates of local convergence on those problems. A quadratic model of the problem requires the solution of a linear system at each iteration, and the evaluation or construction of the matrix in the linear system and the solution of the system typically dominate the solution time. There has been recent interest in using a cubic regularization to the quadratic model when computing the step direction within a nonlinear programming algorithm. The idea itself is not new, in fact, as early as three decades ago, Griewank proposed in [29] a cubic regularization to develop an algorithm based on Newton's method that is affine-invariant and convergent to second-order critical points. More recently, in [38], Nesterov and Polyak re-introduced cubic regularization of Newton's method for unconstrained NLPs and provide global complexity results for certain classes of problems. Their main motivation was that the cubic model is a global upper estimate for the function to be minimized and, therefore, enforces global performance guarantees of the resulting method. Local convergence results were also provided. Further work on cubic regularization was documented by Cartis, Gould, and Toint in [10] and [11], where the authors proposed the Adaptive Regularisation algorithm using Cubics (ARC). This algorithm uses an approximate Hessian and finds an approximate minimizer of the cubic model, which reduces the computational work while retaining the local and global convergence properties of [29] and [38] and the worst-case complexity results of [38].

There has been additional literature [1,2,4–6,18,27,28,30,31,35,36,44] on using cubic regularization since these papers, and many of them propose variations on ARC which uses a trust-region whose radius is linked to the cubic regularization parameter. In ARC, the regularization parameter for the cubic term is initialized at the beginning of the algorithm and then updated at each iteration using a scheme based on sufficient descent. This scheme can have a significant impact on the number of iterations performed by the algorithm. In addition, using the cubic regularization at each iteration of the algorithm means that a nonlinear system will need to be solved at each iteration. As a result, even if the overall iteration count is reduced due to obtaining better step directions, each iteration with a positive definite Hessian will require more computational effort than one iteration with a quadratic model, which requires the solution of a single linear system. In [3], we proposed a hybrid approach that used cubic regularization within the context of Newton's method only during those iterations in which the Hessian is indefinite. We showed that computing the cubic step is equivalent to solving the linear system for a certain value of the Levenberg–Marquardt (LM) perturbation parameter, allowing us to naturally set the cubic regularization parameter in a problem-dependent way for each iteration where it is needed. This selective use of cubic regularization allowed us to use some of its theoretical properties without costing significant extra time for the overall solution method.

In this paper, we focus on the application of a similar hybrid approach in the context of quasi-Newton methods with linesearch. Our philosophy of selectively applying the cubic regularization when the exact Hessian is indefinite within the context of Newton's method can be extended to quasi-Newton methods with Symmetric Rank-One (SR1) updates. SR1 has been shown to outperform popular rank-two updates such as BFGS in terms of the number of iterations and the convergence of Hessian approximation to the true Hessian [9,14]. We chose to investigate SR1 because one of its major drawbacks is that the approximate Hessian may become indefinite, even when (1) is a strictly convex quadratic function (see [43] for a numerical example). Therefore, the extension of our work in [3] to handle indefiniteness via cubic regularization is appropriate for SR1, as well.

The challenge for a straightforward extension is that LM perturbation is a full-rank modification to the approximate Hessian. Since we work with dense matrices with rank-one updates in SR1, it is preferable to work with an approximation to the *inverse* Hessian and update it via a rank-one matrix. Therefore, there is no straightforward way of expressing LM perturbation in the SR1 update formula for the approximation to the inverse Hessian. To overcome this difficulty, we propose using a modified secant equation and incorporate the cubic regularization term into the resulting SR1 update. The resulting approach differs slightly from the classic application of cubic regularization in that the regularization parameter now also has an upper bound in order for the approximation to the inverse Hessian to be positive definite.

The outline of the paper is as follows. In Sect. 2, we briefly review SR1 updates in quasi-Newton methods for solving unconstrained NLPs, and in Sect. 3, we provide a review of adaptive cubic regularization and hybrid cubic regularization as presented in [3]. Using these ideas, we then introduce the Hybrid Cubic Approach for SR1 in Sect. 4, and this new approach uses the original SR1 update when we have a descent direction and the modified update otherwise. In Sect. 5, we provide theoretical guarantees for our proposed method, such as n-step convergence for strictly convex quadratic problems and global convergence under standard assumptions. We have implemented our proposed approach, and in Sect. 6, we present details of our implementation and comparative numerical results.

1.1 Notation

Throughout the paper, $\|a\|$ indicates the Euclidean norm of vector $a \in \mathbb{R}^n$ and $\|A\|$ indicates the Frobenius norm of matrix $A \in \mathbb{R}^{n\times n}$.

2 A review of quasi-Newton methods with SR1 updates

Most quasi-Newton methods with linesearch follow the same general algorithmic framework:

Equation (2) are the sufficient descent and curvature conditions, also known as the strong Wolfe conditions [54,55]. The first condition is also known as the Armijo condition and is used by most quasi-Newton and Newton methods to ensure progress toward a (local) minimum. The second condition is a stronger form of the standard curvature condition, and it ensures that the steplength α is sufficiently large while being an (approximate) minimizer of the function $\phi(\alpha) = f(x_k + \alpha\Delta x)$. The cur-

```
Set $k = 0$ and pick a suitable $x^0$, $\epsilon > 0$, and $0 < \nu_1 < \nu_2 < 1$.
Choose an initial approximate Hessian, $B_0$.
while $\|\nabla f(x^k)\| > \epsilon$ do
    Solve $B_k \Delta x = -\nabla f(x^k)$.

    Find a steplength, $\alpha > 0$, such that

        $f(x^k + \alpha\Delta x) < f(x^k) + \nu_1 \alpha \nabla f(x^k)^T \Delta x$
        $|\nabla f(x^k + \alpha\Delta x)^T \Delta x| < \nu_2 |\nabla f(x^k)^T \Delta x|$.        (2)

    $x^{k+1} \leftarrow x^k + \alpha\Delta x$, $k \leftarrow k + 1$.

    Update $B_k$ using a formula based on the secant equation.
end
```

Algorithm 1: General framework for a quasi-Newton method for solving unconstrained NLPs.

vature condition also ensures that the approximate Hessian remains positive definite for certain rank-two quasi-Newton updates. If a backtracking linesearch is used, the curvature condition may not be necessary, since the linesearch will seek a value of α that satisfies the first condition and is as large as possible.

There are many variants of the update formula: symmetric rank-one (first discovered by [7] and then independently re-discovered by [15,20,37,53]) and rank-two variants such as the Davidon-Fletcher-Powell (DFP) update [16,23], the Broyden–Fletcher–Goldfarb–Shanno (BFGS) update[8,22,26,46]. A unifying framework for many of these updates, including both rank-one and rank-two varieties, was given in [32]. The updates are based on the secant equation

$$y_k = B_k p_k, \tag{3}$$

where $y_k = \nabla f(x^{k+1}) - \nabla f(x^k)$ and $p_k = x^{k+1} - x^k$. The SR1 update formula

$$B_{k+1} = B_k + \frac{(y_k - B_k p_k)(y_k - B_k p_k)^T}{(y_k - B_k p_k)^T p_k} \tag{4}$$

is a rank-one update that satisfies (3) with B_{k+1} in place of B_k. By contrast, the BFGS update formula

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T p_k} - \frac{B_k p_k p_k^T B_k}{p_k^T B_k p_k}$$

is a rank-two update that satisfies (3) with B_{k+1} in place of B_k.

SR1 updates have three major drawbacks:

(1) The denominator of the update term in (4) may vanish, meaning that B_{k+1} is not well-defined.

(2) The step directions obtained using (4) may no longer be independent leading to slow (local) convergence or stalling.
(3) The Hessian approximation may not be positive definite, resulting in a step direction that does not produce descent.

The first drawback is generally remedied by simply skipping the update if the denominator becomes too small, and the details will be given in Sect. 2.5. Skipping an update may prevent fast convergence, but it was shown in [12] and [33] that the denominator vanishes rarely in practice and does not have a significant impact on overall iteration counts or runtimes. The second drawback is also noted in [12] and may impact the convergence of the approximate Hessians to the true Hessian, thereby, affecting the overall performance, especially local convergence, of the algorithm. In fact, it is shown in [33] that many problems do not satisfy the so-called uniform linear independence requirement. Nevertheless, they were able to provide a local convergence proof with only positive definiteness and boundedness assumptions for the approximate Hessian. Therefore, our remedy to the third drawback will address the second drawback as well.

Given these drawbacks, why are we interested in SR1? It is also shown in [33] that, on average, SR1 requires 82–92% of the number of iterations and function evaluations of BFGS on a set of small problems with 2–30 variables. Additionally, [33] and [12] provide computational and theoretical results, respectively, that approximate Hessians generated by the SR1 converge faster to the true Hessian than BFGS, provided that the uniform linear independence assumption is satisfied. In our more recent numerical testing on larger problem sets containing problems with up to 10,000 variables, we have found that SR1 continues to be competitive with or outperform BFGS on problems where the approximate Hessian remains positive definite. Therefore, if the issue of indefiniteness is addressed in a reliable and efficient manner, SR1 can be used in lieu of more popular rank-two updates.

2.1 Approximation of the inverse Hessian

It is also important to note that a rank-one formula for the inverse Hessian approximation exists and can be derived by using the Sherman–Morrison–Woodbury formula [49]. Letting H_k denote the inverse Hessian approximation at iteration k, we have that

$$H_{k+1} = H_k + \frac{(p_k - H_k y_k)(p_k - H_k y_k)^T}{(p_k - H_k y_k)^T y_k} \tag{5}$$

Therefore, in the algorithm above, instead of solving

$$B_k \Delta x = -\nabla f(x^k), \tag{6}$$

we can simply set

$$\Delta x = -H_k \nabla f(x^k), \tag{7}$$

which allows for significant speed up. Of course, this change is only applicable in cases where the inverse exists (H_k is not just the approximation to the inverse Hessian, but also B_k^{-1}), which means that handling the indefiniteness of B_k can also lead to the design of a fast algorithm.

2.2 SR1 update formula derivation

In this section, we will briefly review the steps to derive the SR1 update formula (4), since we will need to revisit these steps in our proposed modified secant equation scheme. There are different ways to derive or justify the SR1 formula, and, here, we follow the steps presented in [21].

We know that B_k aims to satisfy the secant condition (3). However, given that it is an approximation, we have that

$$y_k = B_k p_k + O(\|p_k\|),$$

and our goal is to do better with the next update. That is, we would like B_{k+1} to satisfy the formula

$$y_k = B_{k+1} p_k. \tag{8}$$

We also know that B_{k+1} is obtained after a rank-one update to B_k, and, therefore, has the form

$$B_{k+1} = B_k + \sigma v v^T, \tag{9}$$

where σ is a scalar and $v \in \mathbb{R}^n$.

Substituting this form into the secant equation, we get that

$$y_k = B_k p_k + \sigma \left(v^T p_k\right) v,$$

or, alternatively,

$$y_k - B_k p_k = \sigma \left(v^T p_k\right) v.$$

Since $\sigma \left(v^T p_k\right)$ is a scalar, we can simply set $v = y_k - B_k p_k$ and $\sigma = \left(v^T p_k\right)^{-1}$ to satisfy this equation. Doing so gives us the SR1 update formula (4). Note that the SR1 update formula is unique, that is, there is exactly one rank-one update satisfying the secant equation. By contrast, there are several families of rank-two updates.

2.3 Choosing B_0

In the absence of any information to approximate the Hessian, $B_0 = I$ can be used. Doing so means that the first step of the algorithm will be along the steepest descent direction. Depending on the quality of the initial solution, however, this may not be a good idea. Remedies for the scaling of the matrices in quasi-Newton methods (for any iteration) have been studied in [40,41], and [42], and it is suggested in [48] that these remedies are best used only in the first iteration.

Following [47], we use the identity matrix as merely a trial approximation which helps us gather some information around the initial solution, x_0. We calculate a step direction Δx, a steplength α, and the quantities $\bar{p} = \alpha \Delta x$ and $\bar{y} = \nabla f(x_0 + \alpha \Delta x) - \nabla f(x_0)$. The Hessian approximation is then re-initialized as

$$B_0 = \frac{\bar{y}^T \bar{y}}{\bar{y}^T \bar{p}} I. \tag{10}$$

Note that this re-initialization step is only used when (1) uses a general nonlinear function. For problems where the function being minimized is quadratic, this re-initialization forces the denominator of H_1 to always be 0. To see why this is the case, let's first look at the quantities being used in the re-initialization. (In the following, we assume that

$$f(x) = \frac{1}{2} x^T Q x + c^T x + b$$

with $Q \in \mathbb{R}^{n \times n}$ and symmetric, $c \in \mathbb{R}^n$, and $b \in \mathbb{R}$.)

$$\begin{aligned}
\Delta x &= -I \nabla f(x_0) = -(Qx_0 + c) \\
\bar{p} &= -\alpha(Qx_0 + c) \\
\bar{y} &= (Q(x_0 + \alpha \Delta x) + c) - (Qx_0 + c) = \alpha Q \Delta x = -\alpha Q(Qx_0 + c) \\
B_0 &= \tfrac{\alpha^2 (Qx_0+c)^T QQ(Qx_0+c)}{\alpha^2 (Qx_0+c)^T Q(Qx_0+c)} I = \tfrac{(Qx_0+c)^T QQ(Qx_0+c)}{(Qx_0+c)^T Q(Qx_0+c)} I.
\end{aligned}$$

Then, in the first iteration, we calculate a new Δx and a new α and have:

$$\begin{aligned}
\Delta x &= -B_0^{-1} \nabla f(x_0) = -\tfrac{(Qx_0+c)^T Q(Qx_0+c)}{(Qx_0+c)^T QQ(Qx_0+c)} (Qx_0 + c) \\
p_0 &= \alpha \Delta x = -\alpha \tfrac{(Qx_0+c)^T Q(Qx_0+c)}{(Qx_0+c)^T QQ(Qx_0+c)} (Qx_0 + c) \\
y_0 &= \alpha Q \Delta x = -\alpha \tfrac{(Qx_0+c)^T Q(Qx_0+c)}{(Qx_0+c)^T QQ(Qx_0+c)} Q(Qx_0 + c).
\end{aligned}$$

Using these quantities, it is easy to verify that the denominator of the update to compute H_1, or $(p_0 - {B_0}^{-1} y_0)^T y_0$, vanishes. Therefore, when minimizing a quadratic function, we will simply use $B_0 = I$.

2.4 Undefined updates

As mentioned above, the SR1 update formula (4) becomes undefined if the denominator of the update term vanishes. The denominator vanishes under three possible scenarios:

(1) $p_k = 0$: When $\alpha \Delta x = 0$, we have either converged or stalled. Note that the use of strong Wolfe conditions require that we have a sufficiently large step, which means that in order to have $p_k = 0$ while stalling, the linesearch will have failed.

(2) $y_k - B_k p_k = 0$, $p_k \neq 0$: The secant equation is already satisfied with B_k. Given the derivation in the previous subsection, we keep $B_{k+1} = B_k$ as it is the unique solution to the secant equation.

(3) $y_k - B_k p_k \neq 0$, $p_k \neq 0$, $(y_k - B_k p_k)^T\ p_k = 0$: There is no solution to the secant equation. However, skipping the update has been shown to be an effective remedy both computationally and theoretically, as we still get a descent direction since B_k is positive definite.

Numerically, the denominator of the update does not have to vanish—the iterates can become adversely affected if it simply becomes too small. The update term may start to dominate B_{k+1}, and we lose the valuable information that has been accumulated in B_k (discussed further in the next subsection). To accommodate both the theoretical and the computational concerns, the rule that we have decided to apply is that updates are skipped whenever the denominator is too small in a relative sense:

$$|\,(y_k - B_k p_k)^T\ p_k| < \epsilon_1 \|p_k\| \|y_k - B_k p_k\|, \tag{11}$$

that is, if (11) is satisfied, $B_{k+1} = B_k$. A typical value for ϵ_1 is 10^{-8}. Additionally, we can impose the condition that the update is skipped whenever

$$\frac{\|B_{k+1} - B_k\|}{1 + \|B_k\|} > P, \tag{12}$$

where P is a large constant with a typical value of 10^8.

Since we will be working with H_k rather than B_k in our algorithm, we note that

$$(y_k - B_k p_k)^T\ p_k = -\,(p_k - H_k y_k)^T\ B_k p_k \approx -\,(p_k - H_k y_k)^T\ y_k$$

(with the last term due to (3)), so that whenever (11) is satisfied, we would expect the denominator of the update formula for H_k (5) to vanish as well. Therefore, if

$$\left|(p_k - H_k y_k)^T\ y_k\right| < \epsilon \|y_k\| \|p_k - H_k y_k\|, \tag{13}$$

we set $H_{k+1} = H_k$. Similarly, the update is also skipped whenever

$$\frac{\|H_{k+1} - H_k\|}{1 + \|H_k\|} > P. \tag{14}$$

It should be noted that while skipping updates may promote numerical stability, they may also prevent fast convergence.

2.5 Uniform linear independence

A known problem with SR1 is that it may get stuck in a subspace when subsequent step directions become linearly dependent. A precise definite of uniform linear independence is as follows [12]:

Definition 1 A sequence $\{p_k\}$ is uniformly linearly independent if there exist $\xi > 0$, k_0, and $m \geq n$ such that, for each $k \geq k_0$, we can find n distinct indices $k \leq k_1 \leq k_2 \leq \ldots \leq k_n \leq k + m$ for which the minimum singular value of the matrix $P_k = \left(\frac{p_{k_1}}{\|p_{k_1}\|}, \ldots, \frac{p_{k_n}}{\|p_{k_n}\|} \right)$ is at least ξ.

While the condition is hard to verify in practice for all subsets of directions, it is noted in [12] that a failure is generally observed in practice in the following scenarios:

- When the denominator of the update nearly vanishes
- When solving separable or partially separable problems

The first scenario is purely numerical. When the denominator of the update nearly vanishes, we have that $B_{k+1} \approx \frac{(y_k - B_k p_k)(y_k - B_k p_k)^T}{(y_k - B_k p_k)^T p_k}$, which means that all future approximate Hessians will be dominated by this and other such large updates. Therefore, there will be a subset of step directions calculated in iteration $(k+1)$ or later that will be (nearly) orthogonal to $(y_k - B_k p_k)$ and, therefore, fail to satisfy a uniform linear independence requirement.

The second scenario is inherent to the problem and may not be easily avoided. For separable or partially separable problems, the solution to (1) may be found component-by-component through the iterations, with each additional improvement lying in a different subspace. The step direction components along those subspaces that were handled first may become very small in later iterations and result in a violation of a uniform linear independence requirement.

The importance of a uniform linear independence requirement is that it serves as one of the main assumptions of a proof that the approximate Hessian converges to the true Hessian as the iterates converge to the solution of (1). While the violation of the requirement does not necessarily prevent convergence, it can lead the algorithm to proceed very slowly, even stall. A weaker assumption made by [33] is that the approximate Hessian, B_k, become and remain positive definite, and they show that SR1 is then $2n$-step q-quadratic.

3 A review of adaptive and hybrid cubic regularization

As stated in the introduction, there has recently been a growing body of literature on using cubic regularization within the contexts of Newton and quasi-Newton methods. While many of these papers feature stronger theoretical guarantees for the resulting algorithm, the underlying impact of using cubic regularization is the solution of a nonlinear system of equations in each iteration. The additional work yields better steps in some sense and generally results in a reduction in the number of iterations, but the runtime per iteration is increased. The advantage of an *adaptive* approach is to choose a suitable regularization parameter to ensure good progress at each iteration, but letting the parameter tend to zero while approaching a (local) minimum to take advantage of the fast convergence of Newton's or the quasi-Newton method and reduce the computational burden per iteration. The *hybrid* approach presented in [3], which focuses on Newton's method, takes this one step further and only uses cubic regularization for the iterations where the Hessian is indefinite. Computational studies presented in

[3] show that, not only is there computational justification for selectively applying cubic regularization, but that there is an inherent connection between the remedy for indefiniteness and the cubic regularization parameter. It is this connection that we plan to extend to SR1 in this paper. Therefore, we will use this section to give a brief review of adaptive and hybrid cubic regularization.

3.1 Cubic regularization

The step-direction, Δx, used by Newton's method minimizes

$$f_N(x^k + \Delta x) := f(x^k) + \nabla f(x^k)^T \Delta x + \frac{1}{2}\Delta x^T \nabla^2 f(x) \Delta x. \tag{15}$$

To obtain the cubic regularization formula, we let L be the Lipschitz constant for $\nabla^2 f(x)$. Then, we can also define

$$f_L(x^k + \Delta x) := f(x^k) + \nabla f(x^k)^T \Delta x + \frac{1}{2}\Delta x^T \nabla^2 f(x^k) \Delta x + \frac{L}{6}\|\Delta x\|^3, \tag{16}$$

which has the property that $f_L(x^k + \Delta x) \geq f(x^k + \Delta x)$ for all $\Delta x \in \mathbb{R}^n$. While this gives us a means of building a solution method for (1), we know of no algorithm for explicit determination of L for general unconstrained NLP since finding L is a global optimization problem. Instead, we use an approximation, M, to the Lipschitz constant and define

$$f_M(x^k + \Delta x) := f(x^k) + \nabla f(x^k)^T \Delta x + \frac{1}{2}\Delta x^T \nabla^2 f(x^k) \Delta x + \frac{M}{6}\|\Delta x\|^3. \tag{17}$$

The cubic step direction is found by solving the problem

$$\Delta x \in \arg\min_s f_M(x^k + s). \tag{18}$$

It should also be noted that the Hessian can be replaced by an approximate Hessian in the above functions.

3.2 Adaptive cubic regularization

In [29,38], and [10], it is shown that for sufficiently large M, $x^k + \Delta x$ will satisfy an Armijo condition. On the other hand, $M\|\Delta x\| \to 0$ sufficiently close to a local minimum, so that $f_M(x^k + \Delta x) \approx f_N(x^k + \Delta x)$ and adaptive cubic regularization exhibits fast local convergence similar to Newton's method. Therefore, to ensure sufficient descent at each iteration and to allow for an efficient solution method, we need to control the approximation to the Lipschitz constant, M, rather than impose a steplength α as in Newton's method.

The basic outline of the Adaptive Cubic Regularization method (also known as Adaptive Regularization with Cubics, or ARC) to solve (1) is as follows:

```
Set $k = 0$ and pick a suitable $x^0$, $M > 0, \epsilon > 0, \beta > 1, 0 < \nu < 1, 0 < \chi < 1$.
while $\|\nabla f(x^k)\| > \epsilon$ do
    Find $\Delta x \in \arg\min_s f_M(x^k + s)$.
    if $f(x^k + \Delta x) > f(x^k)$ then
        $M \leftarrow \beta M$.
    else
        if $f(x^k + \Delta x) < f(x^k) + \nu \nabla f(x^k)^T \Delta x$ then $M \leftarrow \chi M$.; $x^{k+1} \leftarrow x^k + \Delta x, k \leftarrow k + 1$.
    end
end
```

Algorithm 2: Basic outline of the adaptive cubic regularization algorithm for unconstrained NLPs as given in [29,38], and [10].

The main step of the algorithm determines Δx, and it is nontrivial as it involves the solution of an unconstrained NLP, albeit one with a special form that can be exploited. In [10], the authors propose solving the optimization problem only approximately and show that the resulting algorithm retains the convergence properties of [38].

Another important issue here is the choice of the regularization parameter, M. A simple approach to initializing M is to start with a very small value and let the first iteration of Algorithm 2 raise it to be sufficiently large. Then, picking β and χ close to 1 will ensure controlled progress. In practice, however, it may be good to be aggressive when decreasing M, especially for well-behaved convex problems, to reduce both iteration counts and the runtime.

3.3 A hybrid approach

Note that the adaptive cubic regularization method outlined in Algorithm 2 starts and proceeds with $M > 0$ at each iteration. However, for certain well-behaved problems and within a small neighborhood of a local minimum where the Hessian is positive definite, the Newton direction suffices to give us a robust and efficient performance. When coupled with the added computational effort of solving (18) instead of a linear system of equations for the Newton direction, it is worthwhile to consider using cubic regularization only on those iterations where we encounter negative curvature.

To motivate this idea, let us examine the solution of (18). Note that the first-order necessary conditions for the optimization problem are

$$\nabla f(x^k) + \left(\nabla^2 f(x^k) + \frac{M}{2}\|s\| I\right) s = 0, \tag{19}$$

and the second-order conditions are

$$\nabla^2 f(x^k) + \frac{M}{2}\left(\frac{ss^T}{\|s\|} + \|s\| I\right) \succeq 0. \tag{20}$$

In the case that $\nabla^2 f(x^k) \succ 0$, we will set $M = 0$, which yields the Newton direction. We will then conduct a linesearch to find a steplength α that will give sufficient descent according to an Armijo condition. By using the Newton direction, we only have to solve a linear system in iteration k.

If we have $\nabla^2 f(x^k) \not\succ 0$, however, the situation is quite different. A popular approach is to use the Levenberg–Marquardt method and replace $\nabla^2 f(x_k)$ with $\nabla^2 f(x_k) + \lambda I$ for a sufficiently large λ. However, determining a suitable value of λ generally requires multiple factorizations, so we may need to expend significant computational effort to do so. Even after doing so, we are not guaranteed to get a good quality step out of this process, and it is possible that a large value for λ will result in a small $\|x\|$ or a small value for λ will result in α close to 0, both of which give too short of a step. Therefore, it would be useful to investigate a cubic regularization step that might require a similar computational effort but guarantee sufficient descent. In that case, the reduction in the number of iterations due to choosing better steps may provide an acceptable trade-off to a slightly increased computational requirement per iteration. A key insight of [3] is that when selectively engaging the cubic step, we do not need to keep a value of M throughout all the iterations of the algorithm, but we can compute a new, data-dependent value of M for each iteration where it will be needed.

```
Set $k = 0$ and pick a suitable $x^0$, $\epsilon > 0$, $0 < \nu < 1$, $\beta, \hat{\beta} > 1$.
while $\|\nabla f(x^k)\| > \epsilon$ do
    if $\nabla^2 f(x^k) \succ 0$ then
        Solve $\nabla^2 f(x^k)\Delta x = -\nabla f(x^k)$.
        Find a steplength, $\alpha$, such that $0 < \alpha \leq 1$, and $f(x^k + \alpha\Delta x) < f(x^k) + \nu\alpha\nabla f(x^k)^T \Delta x$.
    else
        Let $\lambda = -\beta\lambda_{min}(x^k)$.
        Solve $(\nabla^2 f(x^k) + \lambda I)\Delta x = -\nabla f(x^k)$.
        while $f(x^k + \Delta x) > f(x^k) + \nu\nabla f(x^k)^T \Delta x$ do
            $\lambda \leftarrow \hat{\beta}\lambda$.
            Solve $(\nabla^2 f(x^k) + \lambda I)\Delta x = -\nabla f(x^k)$.
        end
    end
    $x^{k+1} \leftarrow x^k + \Delta x, k \leftarrow k + 1$.
end
```

Algorithm 3: Modified Newton's method for solving unconstrained NLPs, with a cubic regularization procedure to handle negative curvature. The cubic regularization used is equivalent to the Levenberg–Marquardt approach for appropriately chosen parameters.

The resulting method is described as Algorithm 3, but it appears to not mention the cubic regularization at all. However, note that the step, Δx, obtained by solving

$$(\nabla^2 f(x^k) + \lambda I)\Delta x = -\nabla f(x^k),$$

satisfies both (19) and (20) when

$$M = \frac{2\lambda}{\|\Delta x\|}.$$

(Further details of the equivalence are provided in [3].) Thus, we simply re-interpret the cubic regularization step as coming from a Levenberg–Marquardt regularization of the Newton step with appropriately related values of λ and M. (A similar insight was mentioned in [52] but not explicitly used.) If this direction does not provide sufficient descent with $\alpha = 1$, then we increase λ and refactor. Since $\|\Delta x\|$ is a strictly decreasing function of λ, M increases with this scheme as well. In this case, we can increase λ conservatively, but M will be increased more aggressively, due to the corresponding decrease in $\|\Delta x\|$. Since the theory presented in [29,38], and [10] guarantees that we can obtain sufficient decrease with a sufficiently large M, our scheme ensures that we will be able to find a value of λ for which a full step will be acceptable.

4 Cubic regularization and symmetric rank-one

In [33], Khalfan, Byrd, and Schnabel recommend using a diagonal perturbation of the form λI whenever B_k obtained via SR1 is indefinite for some iteration k. This matches well with the current practice in Newton's method, as described in the previous section, and it seems to work well in practice per the results provided in [33] on a suite of test problems. As shown in the previous section, this approach is also equivalent to using cubic regularization with a specific choice of M. However, doing so requires that we keep and factor a dense matrix B_k or $B_k + \lambda I$ at each iteration (multiple times if $\lambda > 0$), which is an $O(n^3)$ operation. (There are variants of SR1 where the factors of B_k are updated or a sparsity pattern for B_k is pre-determined, and we leave this as future work.)

In addition to using cubic regularization, we would like to also continue working with H_k rather than B_k. However, finding a suitable perturbation, λ, for B_k and then taking the inverse of $B_k + \lambda I$ is even more costly. We do not know of a concise expression for the inverse of $B_k + \lambda I$, so we have decided to explore integration of cubic regularization into the step direction calculations in a different way than the equivalency to the Levenberg–Marquardt approach. Using a modified secant equation provided one possibility, and we follow the same procedure as Sect. 2.2 to derive our new update formula.

4.1 Modified SR1 update formula derivation

Remember that the secant equation (3) is of the form

$$y_k = Bp_k.$$

This equation is satisfied exactly when B is equal to the true Hessian and the function f in (1) is quadratic. For any other nonlinear function, it is simply an approximation, even when the true Hessian is used. Another interpretation of the secant equation is that it is associated with a quadratic model of the function in the neighborhood of x_k, that is, with f^N.

So what would happen if we were to use cubic regularization on the quadratic approximation, that is, f^M rather than f? We replace the secant equation with

$$y_k^M = Bp_k^M$$

to investigate. Here,

$$\begin{aligned}
p_k^M &= x^{k+1} - x^k = p_k \\
y_k^M &= \nabla f^M(x^{k+1}) - \nabla f^M(x^k) \\
&= \nabla f(x^k) + \nabla^2 f(x_k) p_k + \tfrac{M}{2}\|p_k\| p_k - \nabla f(x^k) \\
&= \left(\nabla f(x^k) + \nabla^2 f(x_k) p_k - \nabla f(x^k)\right) + \tfrac{M}{2}\|p_k\| p_k \\
&\approx \left(\nabla f(x^k) + B_k p_k - \nabla f(x^k)\right) + \tfrac{M}{2}\|p_k\| p_k \\
&= y_k + \tfrac{M}{2}\|p_k\| p_k
\end{aligned}$$

The formula for p_k^M follows from the fact that it simply refers to the difference between two points, which is not impacted by the function being minimized. The formula for y_k^M follows from the fact that $f^M(x^k) = f^M(x^k + 0)$, that is, the cubic regularization formula with $\Delta x = 0$, and the above comment that we are now treating y_k as being associated with the problem of minimizing the quadratic approximation, rather than the function itself.

Note that our goal is still to use cubic regularization *selectively*, which means that we would like to continue to update B_k using a (potentially different) rank-one update. Substituting the above terms into (3), we get

$$y_k = B p_k - \frac{M}{2}\|p_k\| p_k.$$

Thus, our new secant equation can be expressed as

$$y_k = \left(B - \frac{M}{2}\|p_k\| I\right) p_k. \tag{21}$$

Using this new secant equation, we would like to derive a new rank-one update formula for B_k. Following the steps of Sect. 2.2, we note that we need to find γ and u such that

$$B_{k+1} = B_k + \gamma u u^T, \tag{22}$$

where γ is a scalar and $u \in \mathbb{R}^n$.

Substituting this form into the secant equation, we get that

$$y_k = \left(B_k - \frac{M}{2}\|p_k\| I\right) p_k + \gamma \left(u^T p_k\right) u,$$

or, alternatively,

$$y_k - \left(B_k - \frac{M}{2}\|p_k\| I\right) p_k = \gamma \left(u^T p_k\right) u.$$

Since $\gamma\left(u^T p_k\right)$ is a scalar, we can simply set $u = y_k - \left(B_k - \frac{M}{2}\|p_k\| I\right) p_k$ and $\gamma = \left(u^T p_k\right)^{-1}$ to satisfy this equation. Doing so gives us the following modified-SR1 update formula:

$$B_{k+1} = B_k + \frac{\left[y_k - \left(B_k - \frac{M}{2}\|p_k\|I\right) p_k\right]\left[y_k - \left(B_k - \frac{M}{2}\|p_k\|I\right) p_k\right]^T}{\left[y_k - \left(B_k - \frac{M}{2}\|p_k\|I\right) p_k\right]^T p_k} \tag{23}$$

For large values of M, B_{k+1} is dominated by $\frac{M}{2}\frac{p_k p_k^T}{\|p_k\|} \succ 0$. However, since this term is rank-deficient, and we prefer to work with its inverse, which may be numerically unstable for excessively large M, we should be as conservative as possible in our choice of M.

4.2 Updating the inverse

The main purpose of deriving a modified SR1 update is to avoid the storage and factorization of a diagonally perturbed B_k, and instead favoring working directly with H_k. In fact, using a modified secant equation allows us to find the corresponding rank-one update formula for H_k:

$$H_{k+1} = H_k + \frac{\left[p_k - H_k\left(y_k + \frac{M}{2}\|p_k\|p_k\right)\right]\left[p_k - H_k\left(y_k + \frac{M}{2}\|p_k\|p_k\right)\right]^T}{\left[p_k - H_k\left(y_k + \frac{M}{2}\|p_k\|p_k\right)\right]^T\left(y_k + \frac{M}{2}\|p_k\|p_k\right)} \tag{24}$$

The formula is derived by applying the Sherman–Morrison–Woodbury formula [49] to (23) and simplifying.

Assuming the H_k was positive definite and since the numerator of the update is a rank-one positive definite matrix for any value of M including 0, the only way for us to encounter an indefinite H_{k+1} is if the denominator of the fraction in (5) is negative. While it is possible for a negative denominator to also yield a positive definite H_{k+1}, an easy and computationally cheap way to choose M is to use (24) and enforce that the denominator be strictly positive:

$$\left[p_k - H_k\left(y_k + \frac{M}{2}\|p_k\|p_k\right)\right]^T\left(y_k + \frac{M}{2}\|p_k\|p_k\right) > 0.$$

Expanding the left-hand side and regrouping terms, we have that

$$-\frac{\|p_k\|^2 p_k^T H_k p_k}{4}M^2 + \left(\frac{\|p_k\|^3}{2} - \|p_k\|p_k^T H_k y_k\right)M + (p_k - H_k y_k)^T y_k > 0.$$

Letting

$$a = -\frac{\|p_k\|^2 p_k^T H_k p_k}{4}, \qquad b = \frac{\|p_k\|^3}{2} - \|p_k\|p_k^T H_k y_k, \qquad c = (p_k - H_k y_k)^T y_k,$$

we know that $a < 0$ since $H_k \succ 0$. This means that the quadratic $aM^2 + bM + c$ is a concave function and is, therefore, positive-valued between its roots, if they exist. As usual, in order for the roots to exist, we need $b^2 - 4ac \geq 0$.

In addition, we would like to choose $M > 0$. We know that $c < 0$ since c is the denominator of the update for obtaining H_{k+1}, and, as previously mentioned, it must be negative in order for H_{k+1} to fail being positive definite. This implies that $4ac > 0$, which in turn means that if $b > 0$ and $b^2 - 4ac > 0$, both roots of the quadratic,

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad \frac{-b - \sqrt{b^2 - 4ac}}{2a}, \tag{25}$$

are positive. Choosing any value between these two roots for M will ensure that we have a positive definite H_{k+1}. Similarly, if $b < 0$, both roots of the quadratic are negative, which means that we cannot choose a positive value for M.

To summarize, then, we have three cases:

- *Case I* ($b^2 - 4ac < 0$): The quadratic function does not have real roots. This is also the case where the highest value attained by the quadratic (when $M = \frac{-b}{2a}$) is negative, which means that there is no value of M for which we can easily guarantee a positive definite H_{k+1} with this modified SR1 formula.
- *Case II* ($b^2 - 4ac \geq 0$ and $b < 0$): Both roots of the quadratic are nonpositive, which means that we cannot choose a positive value for M.
- *Case III* ($b^2 - 4ac \geq 0$ and $b > 0$): Both roots of the quadratic are nonnegative, so we can pick any M between the roots (25).

In Cases I and II, our approach cannot find a suitable M, which means that we will restart with a multiple of the identity matrix. Doing so is not ideal, as it is equivalent to reverting to steepest descent, which can be quite slow to make progress. However, some alternatives are to have the algorithm fail, to use a rank-two update, or to work with a perturbation of B_{k+1} instead, which can be quite time consuming to handle at each iteration. It is also not unprecedented - even Newton-based methods for nonconvex nonlinear programming with global convergence proofs, such as [34], have used a steepest descent step when negative curvature is encountered.

Our numerical studies show that as long as there are no other numerical issues in the problem, Case III arises more often than Cases I and II, so there is merit in providing a remedy for that case only. As stated before, whenever Case III is encountered, we can pick any value of M between the roots (25). The simple form of $M = \frac{-b}{2a}$ works well, but it also corresponds to the largest value of the denominator as well. In our numerical testing, we have found more success with setting M halfway between the smaller root and this value, so that the quadratic is sufficiently larger than 0 but our choice of M is conservative. Since H_{k+1} retains the modification with M, we want to make sure that we do not deviate too much from our progress toward the true inverse Hessian, which is why it is important to be conservative when making this choice.

It should also be emphasized here that computing the values of a, b, and c, and then choosing M if in Case III is a relatively cheap operation that requires no more additional work than a dot-product, since the matrix-vector multiplication $H_k y_k$ was already formed when using (5) as the update formula. Therefore, applying the cubic regularization in this instance is quite cheap. This is in contrast with the works of [10] where a nonlinear system of equations is solved at each iteration and [3], where a sparse matrix is refactored repeatedly until a suitable value of M is reached.

4.3 Detecting indefiniteness

The usual curvature criterion for quasi-Newton methods, $p_k^T y_k > 0$, is necessary and sufficient for rank-two methods such as BFGS, but it is only a necessary condition for SR1. In fact, we have encountered many instances in which the curvature criterion holds but the direction computed fails to be a descent direction, indicating that the Hessian approximation has become indefinite. Even though it requires additional computational effort, we have found that checking that Δx is a descent direction gives a more reliable indicator of the indefiniteness of H_k.

Of course, there are instances where even this criterion will not detect indefiniteness, that is H_k may be indefinite but still yield a descent direction. This would mean that H_{k+1} computed from H_k may continue to be indefinite, but that if it yields a direction of ascent, it may not be possible to find an M to modify H_{k+1}. In this case, we simply restart the inverse Hessian calculations.

The resulting algorithm is shown as Algorithm 4. Note that we have also adapted the algorithm description to use the inverse Hessian approximation, which emphasizes one of the main advantages of the modified SR1 update approach. The formula for M is derived as discussed above, and we store p_k and $H_k y_k$ from the update and reuse it for the re-update as needed.

5 Convergence of a quasi-Newton method with the modified SR1 update

5.1 *n*-Step quadratic convergence

One of the properties of the SR1 update is n-step convergence on quadratic problems, with the approximate Hessian converging to the true Hessian under some conditions. We show here that this property is maintained for the proposed algorithm.

Theorem 1 *Assume that we have a quadratic function of the form* $\frac{1}{2}x^T Qx + c^T x$ *with a positive definite* Q*. If the step directions calculated using the modified SR1 method are well-defined and* $H_0 - Q^{-1}$ *is positive definite, then the modified SR1 method terminates in at most n iterations. If* $p_1, p_2, \ldots, p_n$ *are linearly independent, we have that* $H^n = Q^{-1}$.

Proof It is well-known that the theorem above, without the definiteness condition, holds for the SR1 method in its original form. The proof can be found in [20]. Additionally, [25] show that under the positive definiteness condition, H_k will remain positive definite for all k, which would then produce a descent direction in that iteration. Therefore, the perturbation never gets triggered, and the modified SR1 method exhibits the same convergence characteristics on strictly convex quadratic problems as the original SR1 method. □

```
Set $k = 0$ and pick a suitable $x^0$, $\epsilon > 0$, and $0 < \nu_1 < \nu_2 < 1$.
Choose an initial approximate Hessian, $B_0$. Let $H_0 = B_0^{-1}$.
while $\|\nabla f(x^k)\| > \epsilon$ do
    Solve $\Delta x = -H_k \nabla f(x^k)$.

    if $\Delta x^T \nabla f(x_k) \geq 0$ then
        Let $M = \frac{-2b+\sqrt{b^2-4ac}}{4a}$.
        Compute $H_{k+1}$ using (24).
    end

    Find a steplength, $\alpha > 0$, such that
        $f(x^k + \alpha\Delta x) < f(x^k) + \nu_1\alpha\nabla f(x^k)^T \Delta x$
        $|\nabla f(x^k + \alpha\Delta x)^T \Delta x| < \nu_2|\nabla f(x^k)^T \Delta x|$.

    $x^{k+1} \leftarrow x^k + \alpha\Delta x$, $k \leftarrow k + 1$.

    Calculate $y_k$ and $p_k$.
    if (13) is satisfied then
        Let $H_{k+1} = H_k$.
    else
        Compute $H_{k+1}$ using (5).
    end
end
```

Algorithm 4: Proposed hybrid SR1 method with the modified SR1 update to incorporate cubic regularization.

5.2 Global convergence

We now provide a proof of convergence to a first-order point under some standard assumptions. Since the modified SR1 algorithm simply reduces to the original SR1 algorithm if no perturbations are triggered, we can apply the convergence results of the original SR1 in that case. Therefore, our focus here is on cases where the perturbation is triggered for at least one iteration. We assume that the set of iterations for which the perturbation is triggered is $\mathcal{P}$.

Two standard assumptions of convergence proofs for quasi-Newton methods are:

A0. f is twice continuously differentiable.

A1. f is bounded below, and the lower level set of x_0 is compact.

We will make these assumptions here as well.

Another standard assumption is that the sequences of Hessian and inverse Hessian approximations remain bounded. Since we have modified the calculation of B_k and H_k for all $k \in \mathcal{P}$, it would not be appropriate to make such an assumption without any investigation.

Lemma 1 *If H_k and $(p_k - H_k y_k)(p_k - H_k y_k)^T$ are bounded above in norm, then H_{k+1} is also bounded above in norm when using the modified SR1 formula.*

Proof Let H_k be bounded above in norm and let H_{k+1} be calculated according to (24) for some value of k. Then, we have the three cases as discussed in the previous section. In the first two cases, H_{k+1} equals a multiple of the identity and is bounded above in norm. In the third case, H_{k+1} is updated using (24) with M chosen such that the denominator of the update strictly greater than 0 (in fact, the value M used is halfway between the value that would give a denominator of 0 and the maximizing value of $\frac{-b}{2a}$, ensuring that the denominator is sufficiently larger than 0). Since the matrix $(p_k - H_k y_k)(p_k - H_k y_k)^T$ is bounded above in norm and $\|p_k\|$ is bounded above due to A1, the numerator of the perturbed update is bounded above as well. Therefore, H_{k+1} is bounded above in norm even with the perturbation applied. □

The boundedness of the sequences of Hessian and inverse Hessian approximations would then extend to the modified SR1 method as well. Using this result, we now show that in every iteration, we choose a step direction that is a descent direction and is bounded in magnitude.

Lemma 2 *For each iteration k, the modified SR1 method finds a descent direction, that is* $\Delta x^T \nabla f(x_k) < 0$. *In addition,* $\|\Delta x\| \leq U$ *for some* $U > 0$.

Proof In iteration k if $\Delta x^T \nabla f(x_k) < 0$ is not satisfied by the step direction generated by (5), we either restart the inverse Hessian calculation using a multiple of the identity or use the modified Hessian update formula (24). If the multiple of the identity is used, we have that $\Delta x^T \nabla f(x_k) = -\kappa \nabla f(x_k)^T \nabla f(x_k)$ for some scaling parameter $\kappa > 0$, and assuming that we are not at a first-order point, the step direction is indeed a descent direction.

If the perturbed update formula (24) is used, then note that the numerator of the update is a rank-one positive semidefinite matrix and that M is chosen so that the denominator is positive. This ensures that the update matrix is positive semidefinite. Without loss of generality, H_k is positive definite, and therefore, H_{k+1} is positive definite. Thus, we have that $\Delta x^T \nabla f(x_k) = -\nabla f(x_k)^T H_{k+1} \nabla f(x_k) < 0$.

The boundedness of Δx by magnitude is established as follows. If we have a restart iteration, then $\Delta x = -\kappa \nabla f(x_k)$, and, therefore, $\|\Delta x\| = \kappa \|\nabla f(x_k)\| < U$ for some U by Assumption A1. If we don't have a restart iteration, $\Delta x = -H_{k+1} \nabla f(x_k)$ with H_{k+1} calculated using either (5) or (24), and we showed in Lemma 1 that the sequence of Hessian matrices remains bounded under standard assumptions. Thus, we have that $\|\Delta x\| = \| - H_{k+1} \nabla f(x_k)\| \leq \|H_{k+1}\| \|\nabla f(x_k)\|$ which is bounded above by some $U > 0$. □

Therefore, we have established the existence of a favorable step direction: one that gives descent while remaining bounded. We now show that in each iteration, we will take nonzero step in this direction.

Lemma 3 *In each iteration k, we will find an* $\alpha > 0$ *along* Δx *such that the strong Wolfe conditions* (2) *are satisfied.*

Proof The existence of such an α is well-established under Assumptions A0 and A1 and Lemma 2. To find α, we have implemented a cubic interpolation with a suitable bracketing and sectioning scheme that is guaranteed to converge to an acceptable point [21]. □

Theorem 2 *If Assumptions A0 and A1 are satisfied,* $\{H_k\}$ *remain bounded in norm, and if* f *is Lipschitz continuous, we have that* $\lim_{k\to\infty} \|\nabla f(x_k)\| = 0$.

Proof Given the assumptions and Lemmas 1, 2, and 3, it is a well-established result (see [39], Sect. 3.2) that the resulting algorithm is globally convergent to a first-order point. □

6 Numerical results

In our numerical testing, we compare the performance of the proposed SR1 method, outlined as Algorithm 4, to BFGS as implemented in ConMin [45]. To provide a fair comparison, we have converted ConMin from Fortran to C using f2c [19] and then used the same initializations, stopping criteria, linesearch, and safeguards to implement the proposed SR1 method. We have used a significantly modified version of loqo's [51] ampl [24] interface and overall framework for both implementations.

Numerical testing was conducted on a laptop running OS X El Capitan (Version 10.11.4) with 8GB of main memory and 2.9 GHz Intel Core i7 processor. The test set consisted of 213 unconstrained problems from the CUTEr [13] test suite, for which the ampl models were obtained from [50]. This set does not represent all the unconstrained problems in the CUTEr collection, as we have limited our experiments to problems with 10,000 or fewer variables and have removed a small number of problems that were unbounded or nondifferentiable at either the user-supplied initial solution or at the optimal solution. Detailed results are provided in the Appendix. We provide a summary and discussion here.

Both codes were given an iteration limit of 10^6, and the maximum number of function evaluations was set at 10^{12}. The proposed SR1 method solved 194 of the 213 problems (91.0%), reached the iteration limit on 8 problems (3.8%), and exited due to a numerical error on 11 problems (5.2%). The numerical errors occurred due to functional evaluation errors, and 6 of the 11 cases were on the *pfit* family of problems, which are quite ill-behaved. The BFGS method solved 189 of the 213 problems (88.7%), reached the iteration limit on 3 problems (1.4%), exited due to a numerical error on 1 problem (0.5%), and exited due to reaching the maximum number of function evaluations on 20 problems (9.4%). The latter issue was observed on a range of problems and always due to the linesearch entering an infinite loop. Since the linesearch implemented in ConMin and used by both codes works by building a cubic model of the function and trying to minimize the cubic model (stopping at any iteration that gives sufficient function value decrease subject to an Armijo condition), one possible remedy would be to switch to a different linesearch technique such as bisection. However, we wanted to keep the integrity of ConMin. It is puzzling, however, that the step directions generated by SR1 do not run into this issue, but any potential reason is outside of the scope of this paper and will be investigated in the future.

In addition to the success of the proposed SR1 method on more problems than BFGS, it is also generally faster. In Fig. 1, we present a performance profile [17] comparing the two codes with respect to runtime on 55 problems where one or both codes took at least 0.1 CPU seconds to reach the optimal solution or had $n \geq 500$ in instances where both codes failed. For runtimes less than 0.1 CPU seconds, we have

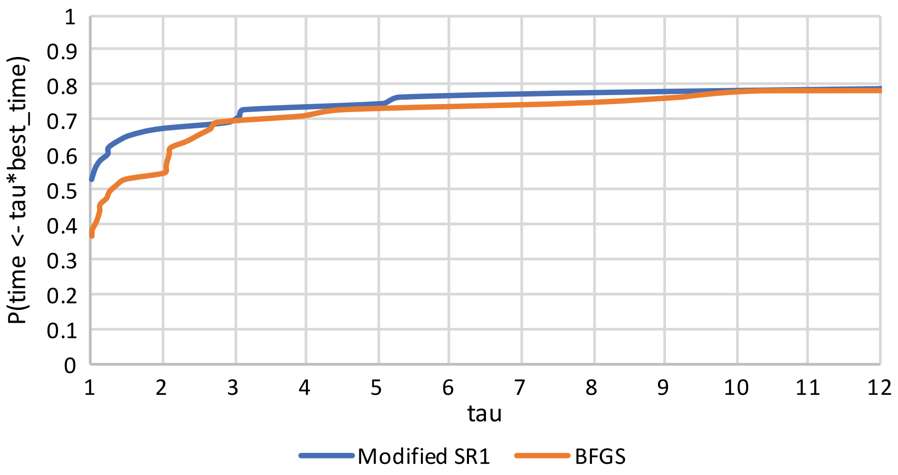

Fig. 1 Performance profiles of the modified SR1 and BFGS codes on unconstrained NLPs. The metric used in this figure is CPU times. The 55 problems included are those where one or both codes took at least 0.1 CPU seconds to reach the optimal solution or had $n \geq 500$ in instances where both codes failed

found that small differences in system performance could have provided an unfair comparison. We chose to include the failures to give a fair comparison of the two codes and observed that the problem size we chose corresponded to cases where the solved problems would take at least 0.1 seconds. It is clear to see from Fig. 1 that SR1 is generally faster than BFGS on this group of problems.

We re-emphasize here that we set M in each iteration where it is needed using a single calculation, rather than an iterative approach. The advantage this has is that the additional computational burden of using the modified SR1 approach is one single update to the inverse Hessian approximation per iteration. In fact, over the 55 problems included in our runtime comparison, the average ratio of the per-iteration-runtime of the SR1 to the BFGS is 1.048, meaning that on average we spend 4.8% more time per iteration with the modified SR1 approach (the median is 1.070). In fact, aside from a small number of problems with numerical issues that may result in significantly different linesearch performances, 80% of the ratios range from 0.9 to 1.2. This means that if the use of the SR1 approach, which is known to produce better approximations to the Hessian matrix than its rank-two counterparts, coupled with the remedy of the hybrid cubic modification for some instances, can lead to a reduction in the iteration counts, the rather low additional computational burden at each iteration can be easily overcome.

In light of this per-iteration-runtime comparison, it may also be useful to look at a comparison of the iteration counts from each solver. We first start with a performance profile comparing iteration counts on the 55 problems considered for Fig. 1. Of these 55 problems, 53 of them had 500 variables or more, and 48 had 1000 variables or more, so we can characterize the group as the large problems. (These 53 problems that have 500 or more variables represent all such problems in the entire test set.) Figure 2 displays the same pattern as Fig. 1 of improvement over the performance of BFGS, this time with respect to iteration counts. Given the problem sizes, we can see that the

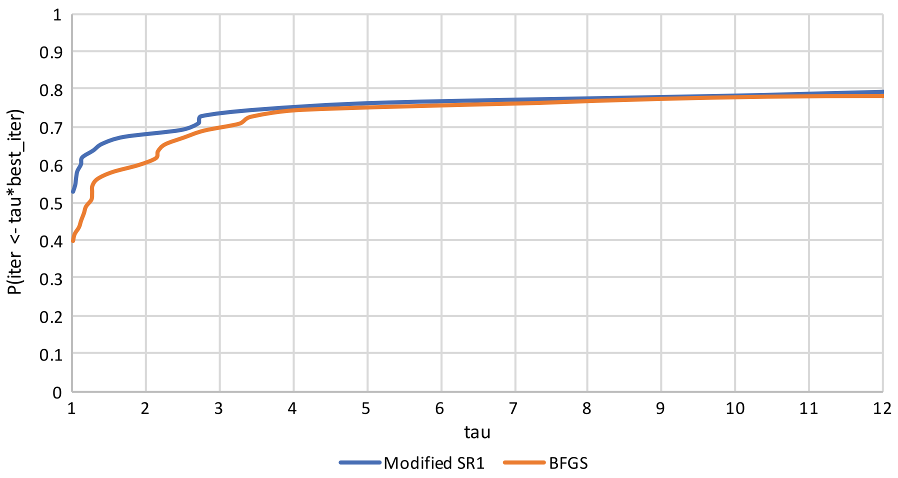

Fig. 2 Performance profiles of the modified SR1 and BFGS codes on unconstrained NLPs. The metric used in this figure is iteration counts. The 55 problems included are those where one or both codes took at least 0.1 CPU seconds to reach the optimal solution or had $n \geq 500$ in instances where both codes failed

modified SR1 approach has improved both the iteration counts and the runtimes over BFGS for the large problems in our test set.

Moreover, basing a comparison on iteration counts allows us to include all problems, rather than just the large ones, in our performance profile. It can be seen from Fig. 3 that the iteration comparisons over the entire test set are relatively even, and, in about 54% of the problems, SR1 has the best or is within 4.8% of the best iteration count (about 55.5% is within 7% of the best). This is consistent with the findings from the runtime analysis on the larger problems as well - SR1 is able to offset the additional computational burden of the cubic regularization in 54–55% of the problems by decreasing the overall number of iterations. Even when performing worse, it is still more consistent than BFGS: about 75% of the cases, SR1 is within 5 times the number of iterations of the best solver, but for the same percentile, BFGS is only within about 8 times the iteration count of the best.

While solving 194 of the 213 problems, the modified SR1 approach skipped updates ((13) was satisfied) 41 times. It encountered an indefinite Hessian approximation (as indicated by obtaining an ascent direction), in 135 of the 194 (70%) problems it solved. The significant number of problems with indefinite Hessian approximations in this standard test set goes to emphasize the need to remedy this issue. On these 135 problems, cubic regularization was used in 60% (on average) of the iterations where a remedy was needed (Case III), and restarts happened the remaining 40% of the time (Cases I and II). We believe that this shows the effectiveness of a simple and cheap remedy, and it also leaves an interesting research question for how to structure these restarts effectively.

We will end with a short discussion on the robustness of the two codes on the "harder" problem instances. These instances, such as the problem families *pfit*, *palmer*, and *dixmaan*, individual problems such as *fminsrf2*, and least-squares reformulations of some feasibility problems with over-fitting issues, tend to cause or were designed

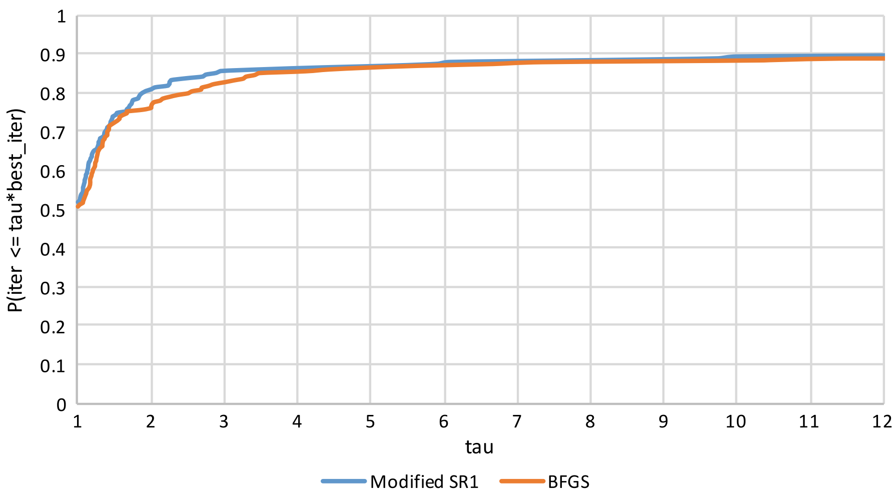

Fig. 3 Performance profiles of the Newton and cubic hybrid codes on unconstrained NLPs. The metric used in this figure is iteration counts. All 213 problems from the test set are included

to cause numerical difficulties for many codes. Of the 177 problems that are solved by both codes, SR1 improves on BFGS' iteration count by at least 1000 iterations on 6 of those instances. On the other hand, BFGS is at least 1000 iterations better on 10 instances, including 1 instance (*noncvxu2*) in which SR1 goes to a solution with a better objective function value. If we look at a ratio of iteration counts instead, we see that SR1 exceeds a ratio of 12 (the largest value in the performance profile) on 4 instances, and BFGS has a maximum ratio of 11.63. Moreover, SR1 solves 17 instances that BFGS does not, and BFGS solves 12 instances that SR1 does not. This suggests that there is a slight difference between the two codes in terms of the outliers (BFGS better on commonly solved problems, SR1 better when at least one solver fails).

It should also be noted that 3 of the 4 instances where SR1 exhibits a significantly higher number and ratio of iterations than BFGS (*msqrtals*, *msqrtbls*, *fminsrf2*), it takes more than 23,000 iterations. A more practical iteration limit for the codes would have meant that SR1 would have been listed as having reached the iteration limit on these three problems, bringing the robustness comparison of the two codes to be relatively even: On commonly solved problems, SR1 would have been better on 6 instances, BFGS better on 7 instances by at least 1000 iterations. SR1 would have exceeded a ratio of 12 only once. SR1 would have failed on 15 problems solved by BFGS, and BFGS would have failed on 17 problems solved by SR1.

7 Conclusion

In this paper, we presented a new SR1 update scheme which uses a modified secant equation derived from the cubic regularization to the quadratic model of a function. The update scheme is applied only on those iterations where a descent direction cannot be obtained using the standard formula. Using the modified secant equation also allows to derive an update formula for the inverse Hessian approximation, and, therefore, we can build our algorithm without needing to invert or factor a dense matrix. Convergence theory and numerical results are provided, showing that the proposed approach is both globally convergent and generally faster than BFGS on a standard test set.

Acknowledgements We would like to thank Daniel Bienstock and Andreas Waechter for their handling of the paper as Editor and Associate Editor, respectively, for MPC. We would also like to thank the two anonymous referees whose feedback and suggestions have greatly improved the paper.

Appendix

See Table 1.

Table 1 Iteration and runtime (in CPU seconds) results for the proposed modified SR1 approach and BFGS on unconstrained NLPs from the CUTEr test set

Problem	n	Modified SR1			BFGS		
		Iters	Time	$f(x*)$	Iters	Time	$f(x*)$
aircrftb	5	47	0.000	0.000	44	0.001	0.000
allinitu	4	12	0.000	5.744	12	0.000	5.744
arglina	100	1	0.001	100.000	1	0.004	100.000
arglinb	10	1	0.000	4.634	2	0.000	4.634
arglinc	8	1	0.000	6.135	2	0.000	6.135
arwhead	5000			(IL)			(LINE)
bard	3	15	0.000	0.008	25	0.000	0.008
bdexp	5000	17	5.241	0.001	17	5.257	0.001
bdqrtic	1000	88	0.755	3983.818			(LINE)
beale	2	15	0.000	0.000	16	0.000	0.000
biggs3	3	24	0.000	0.000	20	0.000	0.000
biggs5	5	112	0.001	0.006	76	0.001	0.006
biggs6	6	88	0.001	0.000	118	0.002	0.000
box2	2	10	0.000	0.000	5	0.000	0.000
box3	3	14	0.000	0.000	13	0.000	0.000
bratu1d	1001			(EVAL)			(LINE)
brkmcc	2	9	0.000	0.169	4	0.000	0.169
brownal	10	10	0.000	0.000	11	0.000	0.000
brownbs	2	13	0.000	0.000	10	0.000	0.000
brownden	4	23	0.001	85,822.202	37	0.001	85,822.202
broydn7d	1000	378	3.472	374.526	297	2.248	365.796
brybnd	5000	27	8.788	0.000	34	10.920	0.000
chainwoo	1000	187	1.596	1.000	469	3.348	4.573
chnrosnb	50	459	0.008	0.000	269	0.005	0.000
cliff	2	3	0.001	0.000			(EVAL)
clplatea	4970	5799	1772.120	−0.013	580	182.254	−0.013
clplateb	4970	1225	431.320	−6.988	447	138.601	−6.988
cosine	10,000			(IL)	14	18.929	−9999.000
cragglvy	5000	88	31.249	1688.215			(LINE)
cube	2	31	0.000	0.000	29	0.000	0.000
deconvu	51	2647	0.105	0.000	86	0.003	0.000

Table 1 continued

Problem	n	Modified SR1			BFGS		
		Iters	Time	$f(x*)$	Iters	Time	$f(x*)$
denschna	2	10	0.000	0.000	12	0.000	0.000
denschnb	2	9	0.000	0.000	9	0.000	0.000
denschnc	2	19	0.000	0.000	22	0.001	0.000
denschnd	3	69	0.000	0.000	92	0.001	0.000
denschne	3	26	0.000	0.000			(EVAL)
denschnf	2	11	0.000	0.000	12	0.000	0.000
dixmaana	3000	9	0.840	1.000	17	1.707	1.000
dixmaanb	3000	13	1.292	1.000	28	2.686	1.000
dixmaanc	3000	16	1.631	1.000	34	3.283	1.000
dixmaand	3000	20	2.110	1.000	45	4.328	1.000
dixmaane	3000	1242	116.429	1.000	1206	109.774	1.000
dixmaanf	3000	368	37.857	1.000	1024	94.188	1.000
dixmaang	3000	1703	150.896	1.000	1217	111.171	1.000
dixmaanh	3000	950	90.330	1.000	1090	100.547	1.000
dixmaani	3000			(IL)	19,146	1830.670	1.000
dixmaanj	3000	3585	318.487	1.000	14,720	1433.890	1.000
dixmaank	3000	15,465	1337.190	1.000	16,719	1609.120	1.000
dixmaanl	3000	2246	199.981	1.000	16,463	1532.540	1.000
dixon3dq	10	18	0.000	0.000	21	0.001	0.000
dqdrtic	5000	8	2.292	0.000	10	3.063	0.000
dqrtic	5000	131	48.048	0.160	427	127.586	0.116
edensch	2000	17	0.702	12,003.285	20	0.761	12,003.285
eg2	1000	6	0.036	−998.947			(LINE)
engval1	5000	25	8.078	5548.668			(LINE)
engval2	3	41	0.000	0.000	27	0.000	0.000
errinros	50	452	0.010	39.904	397	0.007	39.904
expfit	2	13	0.000	0.241	13	0.000	0.241
extrosnb	10	0	0.000	0.000	0	0.000	0.000
fletcbv2	100	807	0.042	−0.514	100	0.004	−0.514
fletchcr	100	749	0.037	0.000	173	0.009	0.000
flosp2hl	650			(IL)			(LINE)
flosp2hm	650			(IL)			(LINE)
flosp2th	650			(IL)			(LINE)
flosp2tm	650			(IL)			(LINE)
fminsrf2	1024	169,458	1232.500	1.000	222	1.753	1.000
fminsurf	1024	1075	8.433	1.000	203	1.590	1.000
freuroth	5000	18	6.300	608,159.189			(LINE)
genrose	500	6055	6.267	1.000	1705	1.236	1.000
growth	3	1	0.000	3542.149			(LINE)
growthls	3	1	0.000	3542.149			(LINE)
gulf	3	47	0.001	0.000	32	0.001	0.000
hairy	2			(EVAL)	54	0.001	20.000
hatfldd	3	26	0.000	0.000	18	0.000	0.000
hatflde	3	34	0.000	0.000	25	0.000	0.000
heart6ls	6	891	0.008	0.000	804	0.008	0.000
heart8ls	8	450	0.004	0.000	2542	0.026	0.000
helix	3	30	0.000	0.000	29	0.001	0.000
hilberta	10	15	0.000	0.000	52	0.001	0.000
hilbertb	50	7	0.000	0.000	7	0.001	0.000

Table 1 continued

Problem	n	Modified SR1			BFGS		
		Iters	Time	$f(x*)$	Iters	Time	$f(x*)$
himmelbb	2	11	0.000	0.000	10	0.000	0.000
himmelbf	4	34	0.000	318.572	29	0.001	318.572
himmelbg	2	8	0.000	0.000	9	0.000	0.000
himmelbh	2	8	0.000	−1.000	7	0.000	−1.000
humps	2	83	0.001	0.000	111	0.002	0.000
jensmp	2	3	0.002	2010.574	20	0.001	124.362
kowosb	4	41	0.001	0.000	32	0.001	0.000
liarwhd	10,000	18	28.060	0.000	20	27.335	0.000
mancino	100	18	0.101	0.000	17	0.106	0.000
maratosb	2	5	0.001	−1.000			(LINE)
methanb8	31	468	0.011	0.000	162	0.006	0.000
methanl8	31	2345	0.058	0.002	386	0.012	0.002
mexhat	2	10	0.000	−0.040	8	0.000	−0.040
minsurf	36	17	0.000	1.000	15	0.001	1.000
morebv	5000	0	0.043	0.000	0	0.169	0.000
msqrtals	1024	51,339	665.522	0.000	1732	17.272	0.000
msqrtbls	1024	23,372	312.256	0.000	1612	16.802	0.000
nasty	2			(EVAL)	3	0.000	0.000
ncb20	1010	136	1.881	1668.907			(LINE)
ncb20b	1000	77	3.885	1721.840			(LINE)
noncvxu2	1000	3861	29.786	2317.180	1426	9.725	2317.749
noncvxun	1000	25	0.201	2316.808	24	0.163	2316.808
nondia	9999	9	1.092	0.000	8	10.092	0.000
nondquar	10,000	422	697.313	0.000	1442	1913.620	0.000
nonmsqrt	9	412	0.002	0.752			(CONV)
osbornea	5	178	0.003	0.000	87	0.002	0.000
osborneb	11	102	0.006	0.040	75	0.004	0.040
palmer1c	8	11	0.000	0.098	114	0.001	0.098
palmer1d	7	48	0.000	0.653			(LINE)
palmer1e	8			(EVAL)	212	0.004	0.001
palmer2c	8	48	0.000	0.014	150	0.001	0.014
palmer2e	8	105	0.002	0.000	211	0.003	0.000
palmer3c	8	27	0.000	0.020	134	0.001	0.020
palmer3e	8	1419	0.013	0.000	240	0.004	0.000
palmer4c	8	23	0.000	0.050	20	0.000	0.050
palmer4e	8	45	0.001	0.000	148	0.002	0.000
palmer5c	6	11	0.000	2.128	15	0.001	2.128
palmer5d	4	7	0.000	87.339	31	0.001	87.339
palmer6c	8	29	0.000	0.016	87	0.001	0.016
palmer7c	8	68	0.000	0.602	93	0.001	0.602
palmer8c	8	33	0.000	0.160	24	0.000	0.160
penalty1	1000	139	1.253	0.010	82	0.642	0.010
penalty2	100	171	0.022	97,096.084			(LINE)
penalty3	100			(EVAL)			(LINE)
pfit1	3			(EVAL)	333	0.004	0.000
pfit1ls	3			(EVAL)	333	0.003	0.000
pfit2	3			(EVAL)	524	0.004	0.000
pfit2ls	3			(EVAL)	524	0.004	0.000
pfit4	3			(EVAL)	955	0.008	0.000
pfit4ls	3			(EVAL)	955	0.009	0.000

Table 1 continued

Problem	n	Modified SR1			BFGS		
		Iters	Time	$f(x*)$	Iters	Time	$f(x*)$
powellsg	4	37	0.000	0.000	58	0.001	0.000
power	1000	5780	47.952	0.000	2342	16.570	0.000
rosenbr	2	44	0.000	0.000	37	0.000	0.000
s201	2	5	0.000	0.000	7	0.000	0.000
s202	2	12	0.000	48.984	18	0.000	48.984
s204	2	7	0.000	0.184	9	0.000	0.184
s205	2	13	0.000	0.000	12	0.000	0.000
s206	2	9	0.000	0.000	8	0.000	0.000
s207	2	13	0.000	0.000	9	0.000	0.000
s208	2	44	0.000	0.000	37	0.000	0.000
s209	2	160	0.001	0.000	123	0.001	0.000
s210	2	257	0.001	0.000	522	0.004	0.000
s211	2	31	0.000	0.000	29	0.001	0.000
s212	2	12	0.000	0.000	14	0.000	0.000
s213	2	49	0.000	0.000	32	0.000	0.000
s240	3	5	0.000	0.000	3	0.000	0.000
s243	3	10	0.000	0.797	13	0.001	0.797
s245	3	14	0.000	0.000	33	0.000	0.000
s246	3	30	0.000	0.000	21	0.000	0.000
s256	4	37	0.000	0.000	58	0.001	0.000
s258	4	26	0.000	0.000	37	0.000	0.000
s260	4	26	0.000	0.000	37	0.001	0.000
s261	4	43	0.000	0.000	48	0.001	0.000
s266	5	14	0.000	1.000	15	0.001	1.000
s267	5	38	0.001	0.000	54	0.001	0.000
s271	6	8	0.000	0.000	8	0.000	0.000
s272	6	48	0.000	0.006	44	0.001	0.006
s272a	6	448	0.006	0.000	46	0.001	0.034
s273	6	22	0.000	0.000	59	0.001	0.000
s274	2	4	0.000	0.000	4	0.000	0.000
s275	4	6	0.000	0.000	7	0.000	0.000
s276	6	6	0.000	0.000	7	0.000	0.000
s281a	10	16	0.000	0.000	43	0.000	0.000
s282	10	362	0.002	0.000	123	0.001	0.000
s283	10	125	0.002	0.000	319	0.003	0.000
s286	20	51	0.000	0.000	35	0.001	0.000
s287	20	26	0.000	0.000	38	0.001	0.000
s288	20	47	0.000	0.000	58	0.001	0.000
s289	30	5	0.000	0.000	3	0.000	0.000
s290	2	5	0.000	0.000	7	0.000	0.000
s291	10	15	0.000	0.000	19	0.000	0.000
s292	30	32	0.000	0.000	39	0.001	0.000
s293	50	44	0.001	0.000	53	0.001	0.000
s294	6	76	0.000	0.000	60	0.001	0.000
s295	10	194	0.001	0.000	87	0.001	0.000
s296	16	231	0.002	0.000	123	0.001	0.000
s297	30	431	0.005	0.000	224	0.002	0.000
s298	50	799	0.016	0.000	352	0.006	0.000
s299	100	1179	0.066	0.000	681	0.032	0.000
s300	20	27	0.000	−20.000	32	0.000	−20.000

Table 1 continued

Problem	n	Modified SR1			BFGS		
		Iters	Time	$f(x*)$	Iters	Time	$f(x*)$
s301	50	56	0.001	−50.000	69	0.002	−50.000
s302	100	106	0.005	−100.000	131	0.007	−100.000
s303	20	11	0.000	0.000	14	0.000	0.000
s304	50	20	0.001	0.000	25	0.001	0.000
s305	100	28	0.001	0.000	24	0.001	0.000
s308	2	15	0.000	0.773	15	0.000	0.773
s309	2	22	0.000	−3.987	12	0.000	−3.987
s311	2	8	0.000	0.000	7	0.000	0.000
s312	2	40	0.000	5.923	33	0.001	5.923
s314	2	9	0.000	0.169	4	0.000	0.169
s333	3	39	0.001	0.043			(LINE)
s334	3	15	0.000	0.008	25	0.000	0.008
s350	4	41	0.000	0.000	32	0.001	0.000
s351	4	37	0.000	318.572	44	0.001	318.572
s352	4	7	0.000	903.234	20	0.001	903.234
s370	6	74	0.001	0.002	53	0.001	0.002
s371	9	226	0.002	0.000	124	0.002	0.000
s379	11	101	0.006	0.040	75	0.004	0.040
s386	2	5	0.000	0.000	7	0.000	0.000
sineval	2			(IL)	66	0.001	0.000
sinquad	10,000	226	389.181	0.000	231	347.267	0.000
sisser	2	12	0.000	0.000	11	0.000	0.000
snail	2	170	0.001	0.000	92	0.001	0.000
srosenbr	10,000	41	65.329	0.000	37	53.320	0.000
testquad	1000	283	2.409	0.000	3291	25.220	0.000
tointgss	10,000	3	2.072	10.001	4	4.810	10.001
tquartic	10,000	19	30.433	0.000	24	33.719	0.000
vardim	100	37	0.003	0.000	36	0.002	0.000
vibrbeam	8	78	0.012	8.481			(LINE)
watson	31	285	0.008	0.000	163	0.003	0.000
woods	10,000	26	39.082	0.000	40	57.429	0.000
yfitu	3	113	0.001	0.000	65	0.001	0.000
zangwil2	2	2	0.000	−18.200	2	0.000	−18.200

(CONV) indicates that BFGS stopped making progress before reaching the required level of convergence, (EVAL) indicates that the code encountered a function evaluation error, (IL) indicates that the code reached its iteration limit, and (LINE) indicates that the line search in BFGS failed

References

1. Anandkumar, A., Ge, R.: Efficient approaches for escaping higher order saddle points in non-convex optimization. (2016). arXiv preprint arXiv:1602.05908
2. Bellavia, S., Morini, B.: Strong local convergence properties of adaptive regularized methods for nonlinear least squares. IMA J. Numer. Anal. **35**, dru021 (2014)
3. Benson, H.Y., Shanno, D.F.: Interior-point methods for nonconvex nonlinear programming: cubic regularization. Comput. Optim. Appl. **58**, 323 (2014)
4. Bianconcini, T., Liuzzi, G., Morini, B., Sciandrone, M.: On the use of iterative methods in cubic regularization for unconstrained optimization. Comput. Optim. Appl. **60**(1), 35–57 (2015)
5. Bianconcini, T., Sciandrone, M.: A cubic regularization algorithm for unconstrained optimization using line search and nonmonotone techniques. Optim. Methods Softw. 1–28 (2016)

6. Birgin, E.G., Gardenghi, J.L., Martınez, J.M., Santos, S.A., Toint, Ph. L.: Worst-case evaluation complexity for unconstrained nonlinear optimization using high-order regularized models. Report naXys-05-2015, University of Namur, Belgium (2015)
7. Broyden, C.G.: Quasi-newton methods and their application to function minimisation. Math. Comput. **21**(99), 368–381 (1967)
8. Broyden, G.C.: The convergence of a class of double-rank minimization algorithms 2. the new algorithm. IMA J. Appl. Math. **6**(3), 222–231 (1970)
9. Byrd, R.H., Khalfan, H.F., Schnabel, R.B.: Analysis of a symmetric rank-one trust region method. SIAM J. Optim. **6**, 1025–1039 (1996)
10. Cartis, C., Gould, N.I.M., Toint, P.L.: Adaptive cubic regularisation methods for unconstrained optimization. Part I: motivation, convergence and numerical results. Math. Program. Ser. A **127**, 245–295 (2011)
11. Cartis, C., Gould, N.I.M., Toint, P.L.: Adaptive cubic regularisation methods for unconstrained optimization. Part II: worst-case function- and derivative-evaluation complexity. Math. Program. Ser. A **130**, 295–319 (2011)
12. Conn, A.R., Gould, N.I.M., Toint, P.L.: Convergence of quasi-newton matrices generated by the symmetric rank one update. Math. Program. **50**(1–3), 177–195 (1991)
13. Conn, A.R., Gould, N., Toint, Ph.L.: Constrained and unconstrained testing environment. http://www.cuter.rl.ac.uk/Problems/mastsif.shtml. Accessed 01 Feb 2018
14. Conn, A.R., Gould, N.I.M., Toint, P.L.: Convergence of quasi-newton matrices generated by the symmetric rank one update. Math. Program. **50**, 177–195 (1991)
15. Davidon, W.C.: Variance algorithm for minimization. Comput. J. **10**(4), 406–410 (1968)
16. Davidon, W.C.: Variable metric method for minimization. SIAM J. Optim. **1**(1), 1–17 (1991)
17. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Technical report, Argonne National Laboratory (2001)
18. Dussault, Jean-Pierre: Simple unified convergence proofs for the trust-region and a new arc variant. Technical report, Technical report, University of Sherbrooke, Sherbrooke, Canada (2015)
19. Feldman, S.I.: A fortran to c converter. In: ACM SIGPLAN Fortran Forum, vol. 9, pp. 21–22. ACM (1990)
20. Fiacco, A.V., McCormick, G.P.: Nonlinear programming: sequential unconstrained minimization techniques. Research Analysis Corporation, McLean Virginia. Republished in 1990 by SIAM, Philadelphia (1968)
21. Fletcher, R.: Practical Methods of Optimization. Wiley, Chichester (1987)
22. Fletcher, R.: A new approach to variable metric algorithms. Comput. J. **13**(3), 317–322 (1970)
23. Fletcher, R., Powell, M.J.D.: A rapidly convergent descent method for minimization. Comput. J. **6**(2), 163–168 (1963)
24. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming. Scientific Press, London (1993)
25. Goldfarb, D.: Sufficient conditions for the convergence of a variable metric algorithm. In: Fletcher, R. (ed.) Optimization, pp. 273–281. Academic Press, New York (1969)
26. Goldfarb, D.: A family of variable-metric methods derived by variational means. Mathematics of computation **24**(109), 23–26 (1970)
27. Göllner, T., Hess, W., Ulbrich, S.: Geometry optimization of branched sheet metal products. PAMM **12**(1), 619–620 (2012)
28. Gould, N.I.M., Porcelli, M., Toint, P.L.: Updating the regularization parameter in the adaptive cubic regularization algorithm. Comput. Optim. Appl. **53**(1), 1–22 (2012)
29. Griewank, A.: The modification of Newton's method for unconstrained optimization by bounding cubic terms. Technical Report NA/12, Department of Applied Mathematics and Theoretical Physics, University of Cambridge (1981)
30. Griewank, A., Fischer, J., Bosse, T.: Cubic overestimation and secant updating for unconstrained optimization of c 2, 1 functions. Optim. Methods Softw. **29**(5), 1075–1089 (2014)
31. Hsia, Y., Sheu, R.-L., Yuan, Y.-X.: On the p-regularized trust region subproblem (2014). arXiv preprint arXiv:1409.4665
32. Huang, H.Y.: Unified approach to quadratically convergent algorithms for function minimization. J. Optim. Theory Appl. **5**(6), 405–423 (1970)
33. Khalfan, H.F., Byrd, R.H., Schnabel, R.B.: A theoretical and experimental study of the symmetric rank-one update. SIAM J. Optim. **3**(1), 1–24 (1993)

34. Liu, X., Sun, J.: Global convergence analysis of line search interior-point methods for nonlinear programming without regularity assumptions. J. Optim. Theory Appl. **125**(3), 609–628 (2005)
35. Sha, L., Wei, Z., Li, L.: A trust region algorithm with adaptive cubic regularization methods for nonsmooth convex minimization. Comput. Optim. Appl. **51**(2), 551–573 (2012)
36. Martınez, J.M., Raydan, M.: Cubic-regularization counterpart of a variable-norm trust-region method for unconstrained minimization. Technical report (2015)
37. Murtagh, B.A., Sargent, R.W.H.: A constrained minimization method with quadratic convergence. Optimization, pp. 215–246 (1969)
38. Nesterov, Y., Polyak, B.T.: Cubic regularization of Newton method and its global performance. Math. Program. Ser. A **108**, 177–205 (2006)
39. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer Series in Operations Research. Springer, Berlin (1999)
40. Oren, S.S., Luenberger, D.G.: Self-scaling variable metric (ssvm) algorithms. Manag. Sci. **20**(5), 863–874 (1974)
41. Oren, S.S., Luenberger, D.G.: Self-scaling variable metric (ssvm) algorithms: Part i: Criteria and sufficient conditions for scaling a class of algorithms. Manag. Sci. **20**(5), 845–862 (1974)
42. Oren, S.S., Spedicato, E.: Optimal conditioning of self-scaling variable metric algorithms. Math. Program. **10**(1), 70–90 (1976)
43. Powell, M.J.D.: Recent advances in unconstrained optimization. Math. Program. **1**, 26–57 (1971)
44. Schiela, A.: A flexible framework for cubic regularization algorithms for non-convex optimization in function space. Technical report (2014)
45. Shanno, D.D., Phua, K.H.: Remark on algorithm 500. minimization of unconstrained multivariate functions. Trans. Math. Softw. **6**(4), 618–622 (1980)
46. Shanno, D.F.: Conditioning of quasi-newton methods for function minimization. Math. Comput. **24**(111), 647–656 (1970)
47. Shanno, D.F.: Conjugate gradient methods with inexact searches. Math. Oper. Res. **3**(3), 244–256 (1978)
48. Shanno, D.F., Phua, K.H.: Matrix conditioning and nonlinear optimization. Math. Program. **14**(1), 149–160 (1978)
49. Sherman, J., Morrison, W.J.: Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix. In: Annals of Mathematical Statistics, volume **20**(4), pp. 621–621. INST MATHEMATICAL STATISTICS IMS BUSINESS OFFICE-SUITE 7, 3401 INVESTMENT BLVD, HAYWARD, CA 94545 (1949)
50. Vanderbei, R.J.: AMPL models. http://orfe.princeton.edu/~rvdb/ampl/nlmodels. Accessed 01 August 2016
51. Vanderbei, R.J., Shanno, D.F.: An interior-point algorithm for nonconvex nonlinear programming. Comput. Optim. Appl. **13**, 231–252 (1999)
52. Weiser, M., Deuflhard, P., Erdmann, B.: Affine conjugate adaptive Newton methods for nonlinear elastomechanics. Optim. Methods Softw. **22**(3), 413–431 (2007)
53. Wolfe, P.: Another variable metric method. Technical report, working paper (1967)
54. Wolfe, P.: Convergence conditions for ascent methods. SIAM Rev. **11**(2), 226–235 (1969)
55. Wolfe, P.: Convergence conditions for ascent methods. ii: Some corrections. SIAM Rev. **13**(2), 185–188 (1971)