Regularized Step Directions in Conjugate Gradient Minimization for Machine Learning

Cassidy K. Buhler
Department of Decision Sciences and MIS, Bennett S. LeBow College of Business, Drexel University, Philadelphia, PA 19104, cb3452@drexel.edu

Hande Y. Benson
Department of Decision Sciences and MIS, Bennett S. LeBow College of Business, Drexel University, Philadelphia, PA 19104 hvb22@drexel.edu

David F. Shanno
RUTCOR (Emeritus), Rutgers University, New Brunswick, NJ 08854 shannod@comcast.net

Conjugate gradient minimization methods (CGM) and their accelerated variants are widely used in machine learning applications. We focus on the use of cubic regularization to improve the CGM direction independent of the steplength (learning rate) computation. Using Shanno's reformulation of CGM as a memoryless BFGS method, we derive new formulas for the regularized step direction, which can be evaluated without additional computational effort. The new step directions are shown to improve iteration counts and runtimes and reduce the need to restart the CGM.

Key words: Conjugate gradient, machine learning, nonlinear programming, unconstrained optimization
History: Submitted on July 20, 2021

1. Introduction

The unconstrained nonlinear programming problem (NLP) has the form

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

where $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$. We assume that $f : \mathbb{R}^n \to \mathbb{R}$ is smooth and its Hessian is Lipschitz continuous on at least the set $\{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$. There are a number of different methods for solving (1) that are usually classified by the amount of derivative information used. For instances where function, gradient, or Hessian evaluations may be unavailable or expensive to evaluate, store, or manipulate, the choice of algorithm may be dictated by what is computationally tractable. Whenever all quantities are readily available, the comparative performance is generally one where there is a trade-off between the number of iterations and the amount of work done per iteration.

In this paper, we focus on conjugate gradient minimization methods (CGM), which are first-order methods for solving (1). These methods are also referred to as *nonlinear conjugate gradient methods* in literature to differentiate them from the classical conjugate gradient methods that were outlined in Hestenes and Stiefel (1952) to solve a linear system of equations. In the context of machine learning, CGM has been successfully used to solve pattern recognition problems (Adeli and Samant (2000), Orozco and García (2003), Khadse et al. (2016a,b)) and in other predictive and classification applications (Saini and Soni (2002), Sözen et al. (2004), Çınar et al. (2009), Azar (2013), Li et al. (2016), Selvamuthu et al. (2019)). It is designed to improve on using only the gradient direction by adding a momentum term: while solving (1), CGM generates a sequence of iterates $\{x_k\}$ such that

$$\begin{aligned} x_{k+1} &= x_k + \alpha_k \Delta x_k \\ \Delta x_{k+1} &= -\nabla f(x_{k+1}) + \beta_k \Delta x_k, \end{aligned} \tag{2}$$

where α_k is the steplength, Δx_k is the step direction, and β_k is a scalar defined as

$$\beta_k = \frac{\nabla f(x_{k+1})^T y_k}{y_k^T \Delta x_k}. \tag{3}$$

Here, we use the standard notation $y_k := \nabla f(x_{k+1}) - \nabla f(x_k)$.

If an exact linesearch is used, under the smoothness assumption, α_k satisfies the first order condition $\Delta x_k^T \nabla f(x_k + \alpha_k \Delta x_k) = 0$, or $\Delta x_k^T \nabla f(x_{k+1}) = 0$. Under the same assumptions, it can also be shown that $\Delta x_k^T \nabla f(x_k) = -\nabla f(x_k)^T \nabla f(x_k)$. Therefore, with exact line search, (3) can be written as

$$\beta_k = \frac{\nabla f(x_{k+1})^T y_k}{\nabla f(x_k)^T \nabla f(x_k)}. \tag{4}$$

This form of β_k gives the Polak-Ribiere formula (Polak and Ribiere (1969)). If f is quadratic, then (4) further reduces to

$$\beta_k = \frac{\nabla f(x_{k+1})^T \nabla f(x_{k+1})}{\nabla f(x_k)^T \nabla f(x_k)}, \tag{5}$$

which is the Fletcher-Reeves formula (Fletcher and Reeves (1964)).

While the momentum term generally yields an improvement over the steepest descent direction, two concerns still remain:

1. The step directions can fail to satisfy the conjugacy condition, and
2. The step length calculation can be a bottleneck for runtime as it requires multiple function evaluations.

The first concern can have a significant impact on the number of iterations to reach the solution of (1). In fact, it is shown in Powell (1976) and Crowder and Wolfe (1972) that CGM as defined by (2) and (3) exhibits a linear rate of convergence unless restarted every n iterations with the steepest descent direction. Moreover, in addition to restarting the method every n iterations, Powell (1977) proposed to use a restart whenever the algorithm moves too far from conjugacy, or more precisely when

$$|\nabla f(x_{k+1})^T \nabla f(x_k)| \geq 0.2 \|\nabla f(x_{k+1})\|^2. \tag{6}$$

In this paper, a restart that occurs after (6) will be called a *Powell restart*. We will show in the numerical results section that nearly all of the problems in our test set require at least one Powell restart and, on average, nearly half of all iterations are Powell restart iterations. Therefore, in order to truly distinguish CGM from steepest descent and improve the rate of convergence, we need a mechanism to improve the step directions. A formal definition of "improvement" will be provided in the next section.

The second concern is about the step length calculation and impacts the amount of effort required per iteration, and, thus, the runtime of the algorithm. An exact line search seeks to find a steplength α^* which solves

$$\min_{\alpha} \Phi(\alpha) = f(x + \alpha \Delta x) \tag{7}$$

for a given point x and step direction Δx. For a general nonlinear function f, this minimization problem in one variable (α) is usually "solved" using an iterative approach such as bisection or cubic interpolation, even though these approaches cannot find the exact value of the minimizing α within a finite number of iterations. For specific forms of f, such as a stricly convex quadratic function, a formula can be used to directly calculate the minimizing α without the need for an iterative approach. We should also note that at the solution of the exact line search, $\Phi'(\alpha) = \nabla f(x + \alpha \Delta x)^T \Delta x = 0$.

By contrast, an inexact line search only seeks to approximately minimize $\Phi(\alpha)$, requiring lower levels of accuracy when the iterates x_k are away from a stationary point of f. In order to maintain theoretical guarantees, most inexact line search techniques rely on guaranteeing *sufficient descent*, that is, $f(x) - f(x + \alpha \Delta x)$ must be sufficiently large according to some criterion. The Armijo criterion is given by

$$f(x + \alpha \Delta x) < f(x) + \epsilon_1 \alpha \nabla f(x)^T \Delta x,$$

and it is accompanied by the curvature condition

$$-\nabla f(x + \alpha \Delta x)^T \Delta x \leq \epsilon_2 \nabla f(x)^T \Delta x,$$

for constants $0 < \epsilon_1 < \epsilon_2 < 1$. The two conditions together are called the *Wolfe conditions*. The curvature condition can also be modified as

$$|\nabla f(x + \alpha \Delta x)^T \Delta x| \leq \epsilon_2 |\nabla f(x)^T \Delta x|$$

to give the *Strong Wolfe conditions*.

While the two best-known forms of CGM are given by (4) and (5) above, they do rely on using an exact line search. This means that when an explicit formula for directly computing the minimizing α is not available, the function f and/or its gradient must be evaluated multiple times for an iterative line search within each iteration of the CGM algorithm. Doing so can be costly if n is large or if the function evaluation is time consuming. For machine learning problems, many gradient-based algorithms choose a fixed step length (also referred to as the learning rate) or use an adaptive approach to setting it. Adagrad (Duchi et al. (2011)), Adadelta (Zeiler (2012)), RMSprop (Geoffrey Hinton and Swersky (2012)), and Adam (Kingma and Ba (2014)) are examples of adaptive learning rate approaches with good performance, but their use does not currently extend to CGM.

In this paper, we propose a cubic regularization variant of CGM, that combines ideas from Benson and Shanno (2014) and Benson and Shanno (2018) to selectively use cubic regularization when solving (1) and from Shanno (1978b) to recast CGM as "memoryless BFGS" and apply an inexact line search. The resulting approach is shown to reduce the need for Powell restarts and to (approximately) optimize the step direction without the typical overload of additional function evaluations. We show in the numerical results that our implementation improves iteration counts and run time on the CUTEr test set (Bongartz et al. (1995)) and on randomly generated machine learning problems. As noted in Griewank (1981) and Benson and Shanno (2014), the approach has an inherent connection to Levenberg-Marquardt's method (Levenberg (1944), Marquardt (1963)) and is, therefore, related to the scaled conjugate gradient method proposed in Møller (1993) for training neural networks. Scaled CGM is the default training function for Matlab's pattern recognition neural network function, `patternnet` (MathWorks (2021)).

In the next section, we introduce the central question of our research and set the vocabulary and notation for the remainder of the paper. Given the extensive body of literature our work pulls from, we believe that clarifying our goals and our scope into a single framework is necessary to clearly communicate our proposed approach and put our results into perspective. We then introduce the cubic regularization of CGM, including a review of the memoryless BFGS formulation proposed by Shanno (Shanno (1978b)), the explicit formulae for the regularized step direction, and the corresponding method for choosing the step length. Unlike other applications of cubic regularization, such as Cartis et al. (2011), Benson and Shanno (2014), Benson and Shanno (2018), the regularized step in our framework can be computed without significant overhead beyond that required for (2) with (3). Moreover, we show that the burden of optimizing the step length can be shifted to optimizing the regularization parameter, which does not require as many function or gradient evaluations as solving (7). In Section 3, we present some theoretical results on computational complexity per iteration, as well as global convergence. Numerical results are given in Section 4.

2. Cubic Regularization for CGM

As discussed, CGM is designed to be an "accelerated" version of steepest descent. The momentum term for the step direction is obtained cheaply so as not to increase the computational burden over the gradient direction, and the local convergence rate is improved from linear to superlinear. However, if the step direction moves away from conjugacy and CGM has to be restarted frequently, CGM will behave more like steepest descent.

The goal of this paper is to propose a regularized method to improve step quality within CGM. Specifically, we aim for this method to

- require fewer iteration counts in computational experiments than its non-regularized version,
- exhibit global convergence with fewer assumptions than its non-regularized version,
- require the same order of computational burden per iteration as its non-regularized version, and
- demonstrate faster overall runtime in computational experiments than its non-regularized version.

In this section, we will present our proposed cubic regularization scheme and its related step length rules. The integration of cubic regularization for CGM will be closer to the approach taken in Benson and Shanno (2014)—we will use the equivalence of cubic regularization to the Levenberg-Marquardt method in order to view it as a regularization of the (approximate) Hessian matrix. (Our previous paper on symmetric rank-1 methods with cubic regularization Benson and Shanno (2018) took the approach of modifying the secant equation, which we are not proposing here but will leave for future work.) Since the formulation of CGM given by (2) and (3) was matrix-free, we will use Shanno's reformulation of CGM as a memoryless BFGS method (Shanno (1978b)). We start with a brief review of the reformulation and then present the proposed new approach.

2.1. Memoryless BFGS Formulation of CGM

It was shown in (Shanno (1978b)) that a version of CGM is equivalent to a memoryless BFGS method, and we will use that equivalence here to build our cubic regularization approach. First, as a reminder and to set notation, the direction is calculated as

$$\Delta x_k = -H_k \nabla f(x_k),$$

where $H_k \approx \left(\nabla^2 f(x_k)\right)^{-1}$, with $H_0 = I$ and H_{k+1} obtained using the BFGS update formula

$$H_{k+1} = H_k - \frac{H_k y_k p_k^T + p_k y_k^T H_k}{p_k^T y_k} + \left(1 + \frac{y_k^T H_k y_k}{p_k^T y_k}\right) \frac{p_k p_k^T}{p_k^T y_k}. \tag{8}$$

Here, y_k is as before and $p_k = \alpha_k \Delta x_k$. A memoryless BFGS method would mean that the updates are not accumulated, that is, H_k is replaced by I in the update formula and

$$H_{k+1} = I - \frac{y_k p_k^T + p_k y_k^T}{p_k^T y_k} + \left(1 + \frac{y_k^T y_k}{p_k^T y_k}\right) \frac{p_k p_k^T}{p_k^T y_k}. \tag{9}$$

To derive the equivalence, Shanno (Shanno (1978b)) notes that Perry (Perry (1978)) expressed (2)-(4) in matrix form as

$$\Delta x_{k+1} = -\left(I - \frac{p_k y_k^T}{p_k^T y_k} + \frac{p_k p_k^T}{p_k^T y_k}\right) \nabla f(x_{k+1}). \tag{10}$$

Shanno (Shanno (1978b)) notes that the matrix in this formulation is not symmetric and adds a further correction:

$$\Delta x_{k+1} = -\left(I - \frac{p_k y_k^T}{y_k^T p_k} - \frac{y_k p_k^T}{y_k^T p_k} + \frac{p_k p_k^T}{p_k^T y_k}\right) \nabla f(x_{k+1}).$$

Finally, to ensure that a secant condition is satisfied, the last term in the matrix is re-scaled:

$$\Delta x_{k+1} = -\left[I - \frac{p_k y_k^T}{y_k^T p_k} - \frac{y_k p_k^T}{y_k^T p_k} + \left(1 + \frac{y_k^T y_k}{p_k^T y_k}\right) \frac{p_k p_k^T}{p_k^T y_k}\right] \nabla f(x_{k+1}). \tag{11}$$

The matrix term is exactly the formula (9) for the memoryless BFGS update.

It is important to note here that, unlike in BFGS, we do not need to store a matrix or a series of updates to calculate Δx_{k+1} using (11). Multiplying $\nabla f(x_{k+1})$ through the matrix in (11) simply requires dot-products, scalar-vector multiplications, and vector addition and subtraction. As such, each update only requires the storage of 3 vectors of length n and $O(n)$ operations.

Furthermore, for an exact line search, (11) reduces to the Polak-Ribiere formula (4), thereby ensuring that our proposed cubic regularization approach remains valid in that case as well. Finally, one advantage of using (11) for CGM is that the criterion $p_k^T y_k > 0$ is always satisfied, which ensures that the sequence of step directions it produces remain stable and is required for most proofs of global convergence, such as the one proposed by Lenard (Lenard (1973)).

2.1.1. Initializations and Restarts In the first iteration, CGM can be initialized using the gradient. However, a two-step process based on the self-scaling proposed in (Oren and Spedicato (1976)) has demonstrated better stability and improved iteration counts. We will use the same initialization scheme so that

$$\begin{aligned} H_0 &= I \\ H_1 &= \frac{p_0^T y_0}{y_0^T y_0}\left(I - \frac{p_0 y_0^T + y_0 p_0^T}{p_0^T y_0} + \frac{y_0^T y_0}{p_0^T y_0}\frac{p_0 p_0^T}{p_0^T y_0}\right) + \frac{p_0 p_0^T}{p_0^T y_0}. \end{aligned} \tag{12}$$

As discussed, CGM is restarted every n iterations (Beale restart) and when condition (6) is satisfied (Powell restart). The Hessian approximation at the most recent restart iteration t is given by

$$H_t = \frac{p_t^T y_t}{y_t^T y_t}\left(I - \frac{p_t y_t^T + y_t p_t^T}{p_t^T y_t} + \frac{y_t^T y_t}{p_t^T y_t}\frac{p_t p_t^T}{p_t^T y_t}\right) + \frac{p_t p_t^T}{p_t^T y_t}, \tag{13}$$

which matches the initialization process given by (12).

We incorporate these changes into our approach by replacing H_k in (8) with H_t as given in (13) instead of I. As such, the formula for the step direction is also modified from (11) to

$$\Delta x_{k+1} = -\left[H_t - \frac{H_t y_k p_k^T + p_k y_k^T H_t}{p_k^T y_k} + \left(1 + \frac{y_k^T H_t y_k}{p_k^T y_k}\right)\frac{p_k p_k^T}{p_k^T y_k}\right]\nabla f(x_{k+1}). \tag{14}$$

Note that $H_t \nabla f(x_{k+1})$ and $H_t y_k$ can be obtained via dot products and scalar-vector multiplications, so there is still no need to compute or store a matrix. With these modifications, the CGM algorithm can be fully described as Algorithm 1.

```
Pick a suitable $x_0$ and $\epsilon > 0$.
Let $\Delta x = -\nabla f(x_0)$, choose $\alpha$ to solve (7), and let $x_1 = x_0 + \alpha \Delta x$.
Set $k = 1$ and $t = 1$.
while $\|\nabla f(x^k)\| > \epsilon$ do
    if $(k - t) \mod n = 0$ (Beale restart) OR (6) is satisfied (Powell restart) then
        $t \leftarrow k$
        $\Delta x \leftarrow -H_k \nabla(x_k)$, where $H_k$ is defined by (13).
    else
        $\Delta x \leftarrow -H_k \nabla(x_k)$, where $H_k$ is defined by (14).
    end
    Choose $\alpha$ to solve (7) given $x_k$ and $\Delta x$.
    $x_{k+1} \leftarrow x_k + \alpha \Delta x$.
    $k \leftarrow k + 1$.
end
```

Algorithm 1: The memoryless BFGS reformulation of CGM, as given by Shanno (1978b).

2.2. Cubic Regularization for Quasi-Newton Methods

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

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

where $B_k \approx \nabla^2 f(x_k)$ and $H_k = B_k^{-1}$.

To obtain the cubic regularization formula, we also define

$$f_M(x_k + \Delta x) := f(x_k) + \nabla f(x_k)^T \Delta x + \frac{1}{2}\Delta x^T B_k \Delta x + \frac{M}{6}\|\Delta x\|^3, \tag{16}$$

where M is the approximation to the Lipschitz constant for $\nabla^2 f(x)$. The cubic step direction is found by solving the problem

$$\Delta x \in \arg\min_s f_M(x^k + s). \tag{17}$$

In Griewank (1981), Nesterov and Polyak (2006), and Cartis et al. (2011), it is shown that for sufficiently large M, $x_k + \Delta x$ will satisfy an Armijo condition and that a line search is not needed when using cubic regularization within a quasi-Newton method or Newton's method. In this new framework, we need to control M rather than α. In Cartis et al. (2011), the ARC method starts with a sufficiently large value of M that is decreased through the iterations and approaches (or is set to) 0 in a neighborhood of the solution. In Benson and Shanno (2014) and Benson and Shanno (2018), the authors proposed setting $M = 0$ for all iterations where the Hessian or its estimate are positive definite and picking a value of M using iteration-specific data only as needed.

In order to motivate the selective use of cubic regularization, we observe that determining Δx using (16)-(17) is nontrivial as it involves the solution of an unconstrained NLP. In Cartis et al. (2011), the authors show that it suffices to solve (17) only approximately in order to achieve global convergence.

Moreover, the use of cubic regularization during iterations with negative curvature is based on its equivalence to the Levenberg-Marquardt method. To see the equivalence, let us examine the solution of (17). Note that the first-order necessary conditions for the optimization problem are

$$\nabla f(x_k) + \left(B_k + \frac{M}{2}\|s\| I\right) s = 0. \tag{18}$$

Similarly, the Levenberg-Marquardt method replaces B_k with $B_k + \lambda I$ for a sufficiently large $\lambda > 0$ to satisfy certain descent criteria. (A Levenberg-Marquardt-based variant of CGM was proposed by Møller (1993) as scaled conjugate gradient methods and remains a popular approach for training neural networks.) Note that the step, Δx, obtained by solving

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

satisfies (18) when

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

(Further details of the equivalence are provided in Benson and Shanno (2014).) Thus, we simply re-interpret the cubic regularization step as coming from a Levenberg-Marquardt regularization of the CGM step with appropriately related values of λ and M. (A similar insight was mentioned in Weiser et al. (2007) but not explicitly used.) If this direction is not accepted, then we increase λ.

2.3. Cubic Regularization for CGM

We posit here that the theoretical difficulties encountered by CGM can be similarly addressed via the cubic regularization approach, without reducing and potentially even improving its overall solution time. In this section, we start by deriving the update formula for an iteration during which cubic regularization is used. Then, we will show the impact of using cubic regularization to reduce the need for restarts in the algorithm. In Section 4, our numerical results show the effectiveness of this approach in practice.

As discussed in the previous section, the use of cubic regularization necessitates that we add a term to the approximate Hessian, that is compute the step direction Δx_k using $B_{k+1} + \lambda I$, instead of B_{k+1}. While the regularization is applied to the approximate Hessian itself, the CGM update formula (8) updates the inverse of the approximate Hessian, that is $H_{k+1} = (B_{k+1})^{-1}$. As such, in order to compute the regularized step direction, we need to compute $(B_{k+1} + \lambda I)^{-1}$. In previous papers that use regularization with BFGS or with Newton's Method, there is no direct formula for computing this inverse. As such, the solution of multiple linear systems may be required at each iteration until a suitable λ value is found, which means that despite improved step directions that reduce the number of iterations, the effort and time per iteration increase, potentially increasing overall solution time as well.

When applying the regularization to the CG update formula, however, we can derive an explicit formula for $(B_{k+1} + \lambda I)^{-1}$. This is a significant advantage for CGM, in that the use of cubic regularization does not incur additional computational burden at each iteration.

We start by showing the formulae for B_t and B_{k+1}.

$$B_t = \left(\hat{H}_t^{-1}\right) = \left(\frac{y_t^T y_t}{p_t^T y_t}\right)\left(I - \frac{p_t p_t^T}{p_t^T p_t} + \frac{y_t y_t^T}{y_t^T y_t}\right) \tag{19}$$

$$B_{k+1} = \left(\hat{H}_{k+1}^{-1}\right) = B_t - \frac{B_t p_k p_k^T B_t}{p_k^T B_t p_k} + \frac{y_k y_k^T}{p_k^T y_k} \tag{20}$$

Next, we will apply the regularization to B_{k+1} and take its inverse. To do so, we will first need to compute the inverse of $B_t + \lambda I$ (henceforth referred to as $H_t(\lambda)$):

$$\begin{aligned} H_t(\lambda) &= (B_t + \lambda I)^{-1} \\ &= \tfrac{p_t^T y_t}{c} I + \tfrac{ab}{c(\lambda b + a)} p_t p_t^T - \tfrac{\lambda}{c(\lambda b + a)} y_t y_t^T - \tfrac{a}{c(\lambda b + a)} (p_t y_t^T + y_t p_t^T), \end{aligned} \tag{21}$$

where

$$a = \frac{y_t^T y_t}{p_t^T p_p}, \quad b = 2\frac{y_t^T y_t}{p_t^T y_t} + \lambda, \quad c = y_t^T y_t + \lambda p_t^T y.$$

It is easy to verify that when $\lambda = 0$, (21) reduces to (13).

We can finally write the formula for computing the inverse of $B_{k+1} + \lambda I$ (henceforth referred to as H_{k+1}):

$$\begin{aligned} H_{k+1}(\lambda) &= (B_{k+1} + \lambda I)^{-1} \\ &= \left(B_t + \lambda I - \tfrac{B_t p_k p_k^T B_t}{p_k^T B_t p_k} + \tfrac{y_k y_k^T}{p_k^T y_k}\right)^{-1} \\ &= (B_t + \lambda I)^{-1} \\ &\quad + \tfrac{p_k^T y_k + y_k^T (B_t + \lambda I)^{-1} y_k}{d} (B_t + \lambda I)^{-1} (B_t p_k)(B_t p_k)^T (B_t + \lambda I)^{-1} \\ &\quad - \tfrac{p_k^T B_t p_k - (B_t p_k)^T (B_t + \lambda I)^{-1} B_t p_k}{d} (B_t + \lambda I)^{-1} y_k y_k^T (B_t + \lambda I)^{-1} \\ &\quad - \tfrac{(B_t p_k)^T (B_t + \lambda I)^{-1} y_k}{d} (B_t + \lambda I)^{-1} (B_t p_k) y_k^T (B_t + \lambda I)^{-1} \\ &\quad - \tfrac{y_k^T (B_t + \lambda I)^{-1} B_t p_k}{d} (B_t + \lambda I)^{-1} y_k (B_t p_k)^T (B_t + \lambda I)^{-1}, \end{aligned} \tag{22}$$

where the denominator d is given by

$$\begin{aligned} d = {} & (p_k^T y_k + y_k^T (B_t + \lambda I)^{-1} y_k)(p_k^T B_t p_k - (B_t p_k)^T (B_t + \lambda I)^{-1} (B_t p_k)) \\ & + ((B_t p_k)^T (B_t + \lambda I)^{-1} y_k)^2. \end{aligned}$$

In order to see that the formula (22) consists of a sum of rank-1 updates to $H_t(\lambda)$, we introduce the following intermediate calculation:

$$\tilde{p}_k = (B_t + \lambda I)^{-1} B_t p_k = \frac{y_t^T y_t}{c} p_k + \frac{\lambda a b p_t^T p_k}{c(\lambda b + a)} p_t - \frac{\lambda^2 y_t^T p_k}{c(\lambda b + a)} y_t - \frac{\lambda a y_t^T p_k}{c(\lambda b + a)} p_t - \frac{\lambda a p_t^T p_k}{c(\lambda b + a)} y_t$$

and rewrite (22) as

$$\begin{aligned} H_{k+1}(\lambda) = {} & H_t(\lambda) - \frac{\tilde{p}_k^T y_k}{d} \left(\tilde{p}_k y_k^T H_t(\lambda) + H_t(\lambda) y_k \tilde{p}_k^T \right) \\ & + \frac{p_k^T y_k + y_k^T H_t(\lambda) y_k}{d} \tilde{p}_k \tilde{p}_k^T \\ & - \frac{p_k^T B_t p_k - p_k^T B_t \tilde{p}_k}{d} H_t(\lambda) y_k y_k^T H_t(\lambda), \end{aligned} \tag{23}$$

and d as

$$d = (p_k^T y_k + y_k^T H_t(\lambda) y_k)(p_k^T B_t p_k - p_k^T B_t \tilde{p}_k) + (y_k^T \tilde{p}_k)^2.$$

When $\lambda = 0$, we have that $\tilde{p}_k = p_k$ and, therefore, (22) reduces to (8).

2.4. Setting a Value for λ

Now that we know how to calculate a step direction with cubic regularization, we need to answer two questions:

1. When do we apply cubic regularization?
2. When applying cubic regularization, how do we choose a value for λ?

The answer to the first question determines how we answer the second one. As shown above, the computation of the regularized step direction does not require significantly more effort than the unregularized one, so it remains to be seen whether being selective with when to apply the regularization is as important for CGM as it was for Newton's method and quasi-Newton methods.

To start with, we decided to try finding a pair (λ, α) which minimized $f(x_k + \alpha \Delta x)$, where Δx was calculated from a regularized step in every iteration. Our preliminary numerical studies showed that this approach reduced the number of iterations for many of the problems, but it significantly worsened the computational effort per iteration by requiring an iterative approach that simultaneously optimized over two variables.

Instead, using Benson and Shanno (2012) as a guide, we pursued the following approach: selectively use cubic regularization whenever the unregularized step direction failed to satisfy the Powell criterion, was not a descent direction, or resulted in a line search failure. The assumption here is that cubic regularization "improves" the step direction in some sense, so it should be deployed when the step direction needs such improvement. In our numerical studies, there were few to no instances of failure to obtain a descent direction or line search failure, so we could not reliably assess the impact of cubic regularization. However, the prevalence of Powell restarts, as will be noted in Section 4, provided a good opportunity to test potential improvements.

When the unregularized step leads to a Powell restart, we will set $\lambda > 0$ and try a regularized step, with increasing values of λ until it no longer results in a Powell restart. The initial value of λ for a Powell restart is computed as

$$\lambda = 5\frac{|\nabla f(x_{k+1})^T \nabla f(x_k)|}{\|\nabla f(x_{k+1})\|^2}. \tag{24}$$

and doubled as needed. For each value of λ, we will choose a corresponding optimal α.

With all the details complete, we now describe the approach, called *Hybrid Cubic Regularization of CGM*, as Algorithm 2.

3. Theoretical Results

As stated at the beginning of Section 2, we had four dimensions to our goal of improving step quality within CGM. Two of those goals were theoretical in nature:

- exhibit global convergence with fewer assumptions than its non-regularized version.
- require the same order of computational burden per iteration as its non-regularized version

In this section, we will demonstrate that we have achieved both of these goals. Since we have mentioned the second goal already, we will start with formalizing it first.

3.1. Computational Burden per Iteration

We start by showing that the while-loop in the cubic regularization steps will be executed a finite number of times. First, we need to ensure that as λ increases, the step direction becomes and/or remains a descent direction.

LEMMA 1. *There exists a finite $\tau_1 > 0$ such that for every $\lambda > \tau_1$, $\Delta x(\lambda)$ is a descent direction.*

Proof. We know that if $H_k(\lambda)$ is positive definite, then $\Delta x(\lambda) = -H_k(\lambda)\nabla f(x_k)$ is a descent direction. Since $H_k(\lambda) = (B_k + \lambda I)^{-1}$, it is guaranteed that there is a value τ_1 such that for all $\lambda > \tau_1$, $B_k + \lambda I$ is positive definite. Therefore, for all $\lambda > \tau_1$, $H_k(\lambda)$ is positive definite as well.

LEMMA 2. *There exists a finite $\tau_2 > 0$ such that for every $\lambda > \tau_2$,* (6) *is satisfied.*

Proof. It is well-established that as $\lambda \to \infty$, the matrix $B_k + \lambda I$ becomes diagonally dominant, and the resulting direction Δx approaches the gradient descent direction. It should be noted that as $\lambda \to \infty$, $\|\Delta x\| \to 0$, but this decrease is offset by the increase in the steplength as the solution to (7). As such, as $\lambda \to \infty$, we have that $|\nabla f(x_{k+1})^T \nabla f(x_k)| \to 0$. This ensures that there is a finite $\tau_2 > 0$ such that for every $\lambda > \tau_2$, (6) is satisfied.

```
Pick a suitable $x_0$ and $\epsilon > 0$.
Let $\Delta x = -\nabla f(x_0)$, choose $\alpha$ to solve (7), and let $x_1 = x_0 + \alpha\Delta x$.
Set $k = 1$ and $t = 1$.
while $\|\nabla f(x^k)\| > \epsilon$ do
    if $(k - t) \mod n = 0$ (Beale restart) then
        $t \leftarrow k$
        $\Delta x \leftarrow -H_k \nabla(x_k)$, where $H_k$ is defined by (13).
        Choose $\alpha$ to solve (7) given $x_k$ and $\Delta x$.
    else
        if (6) is satisfied (Cubic Regularization) then
            Reset $k \leftarrow k - 1$ and initialize $\lambda$ using (24).
            $\Delta x \leftarrow -H_k \nabla(x_k)$, where $H_k$ is defined by (23).
            Choose $\alpha$ to solve (7) given $x_k$ and $\Delta x$.
            while (6) is satisfied do
                $\lambda \leftarrow 2\lambda$.
                $\Delta x \leftarrow -H_k \nabla(x_k)$, where $H_k$ is defined by (23).
                Choose $\alpha$ to solve (7) given $x_k$ and $\Delta x$.
            end
        else
            $\Delta x \leftarrow -H_k \nabla(x_k)$, where $H_k$ is defined by (14).
            Choose $\alpha$ to solve (7) given $x_k$ and $\Delta x$.
        end
    end
    $x_{k+1} \leftarrow x_k + \alpha\Delta x$.
    $k \leftarrow k + 1$.
end
```

Algorithm 2: The hybrid cubic regularization of CGM as described in Section 2.3.

This lemma means that the while-loop in the cubic regularization step will be executed a finite number of times. We now combine this information with our previous analysis of the computational effort required to compute $\Delta x = -H_{k+1}\nabla f(x_k)$.

THEOREM 1. *Computational effort per iteration for Algorithm 2 is of the same order as the computational effort per iteration for Algorithm 1.*

Proof. The Beale restart and the unregularized steps in Algorithm 2 match those of Algorithm 1. Therefore, we only need to analyze the steps with cubic regularization. By Lemmas 1 and 2, we know that we will need to try a finite number of λ values to exit the while-loop in the cubic regularization step with a descent direction that satisfies (6). We can also see that all of the components required to compute Δx with cubic regularization using (23), that is $H_t(\lambda)$, $\tilde{p}_k$, B_t, and d, can all be obtained using vector-vector and scalar-vector operations of the form that does not necessitate the calculation or storage of a matrix to obtain $-H_{k+1}\nabla f(x_k)$. The vectors used in these calculations are the same ones as before (y_t, p_t, y_k, and p_k), which means that the computational burden of the new approach remains the same as in Algorithm 1.

3.2. Global Convergence

We now show that Algorithm 2 is globally convergent. If the Powell restart/cubic regularization step is not invoked, Algorithm 2 is equivalent to Algorithm 1, so in that case, we will call on the convergence results shown in (Shanno (1978a)). Therefore, it suffices to analyze the iterations with cubic regularization.

The key feature of the convergence result in (Shanno (1978a)) is that the condition number of H_k remains bounded above under mild assumptions. Since the only change to the algorithm is to replace H_k with $H_k(\lambda) = (B_k + \lambda I)^{-1}$ in the cubic regularization step, we need to only examine the condition number of $H_k(\lambda)$ in order to use the rest of the proofs given in (Shanno (1978a)). Before we do so, it is important for us to note that the two main assumptions in (Shanno (1978a)) are that the objective function remains bounded below and that the Hessian estimate remains positive definite.

LEMMA 3. *Let us denote the condition number of a matrix W by $\kappa(W)$. Then, for $\lambda > 0$,*

$$\kappa((W^{-1} + \lambda I)^{-1}) \le \kappa(W).$$

Proof. Let χ_{max} and χ_{min} be the maximum and minimum eigenvalues of W, respectively. By the assumptions made in (Shanno (1978a)), we know that $\chi_{min} > 0$. Then,

$$\kappa((W^{-1} + \lambda I)^{-1}) = \kappa(W^{-1} + \lambda I) = \frac{(1/\chi_{min}) + \lambda}{(1/\chi_{max}) + \lambda} = \frac{\chi_{max} + \lambda \chi_{min}\chi_{max}}{\chi_{min} + \lambda \chi_{min}\chi_{max}} \le \frac{\chi_{max}}{\chi_{min}} = \kappa(W).$$

While Algorithm 2 only invokes cubic regularization for a Powell restart, it can be modified to include more cases, such as when the search direction fails to be a descent direction, as another reason to invoke it. Doing so would mean that the assumption of the Hessian estimate remaining positive definite is no longer necessary, as also evidenced by Lemma 1.

With Lemma 3 and the comment above, we can invoke Theorem 7 of (Shanno (1978a)) to conclude that Algorithm 2 exhibits global convergence.

4. Numerical Results

We start this section by describing our testing environment and then we will introduce our numerical results on general unconstrained NLPs from the CUTEr test set (Bongartz et al. (1995)) and two types of machine learning problems with randomly generated data.

4.1. Software and hardware

The original conjugate gradient method described in Algorithm 1 was implemented by Shanno and Phua in the code Conmin using Fortran4 (Shanno and Phua (1976)). We have reimplemented conjugate gradient method of Conmin in C and connected it to AMPL (Fourer et al. (1993)). In our C implementation, we have omitted the BFGS method also implemented in the original Conmin distribution, and we will call our new code Conmin-CG. The software is available for open source download and use (Buhler (2021)). The cubic regularization scheme proposed in this paper as Algorithm 2 was implemented and tested by modifying this software and is also available at the same link.

Machine learning problems presented an opportunity to compare our method to existing software, and we have chosen ADMM (Boyd et al. (2011a)) as the state-of-the-art for our comparison. The Matlab codes that generate the random problem instances and implement ADMM for the Huber function and for group LASSO were obtained from (Boyd et al. (2011b)). In order to provide a fair comparison to ADMM, we implemented Conmin-CG with and without cubic regularization in Matlab as well. The Matlab routines are also available from (Buhler (2021)).

In our numerical testing, we used AMPL Version 20210226 and MATLAB R2021a on a Mac running macOS Catalina Version 10.15.7, with a 2.9 GHz Intel Core i7 processor and 8 GB 1600 MHz DDR3 memory.

4.2. Test set

The test problems were compiled from the CUTEr test set (Bongartz et al. (1995)) as implemented in AMPL (Vanderbei (1997)). We chose 230 unconstrained problems, which included all of the unconstrained problems available from (Vanderbei (1997)) except for those where the objective function could not be evaluated at the provided or default initial solution, where the initial solution was a stationary point (0 iterations), or where the objective function was unbounded. There are 38 QPs and 192 NLPs in the set. We will focus on the two groups of problems separately in the analysis below.

We have included additional problems from machine learning in our test set. These problems were chosen to satisfy the differentiability requirement for our algorithms. (Subgradient-based approaches for nondifferentiable problems will be considered in future work.) The first class of problems use the Huber loss function as follows:

$$\min_x \sum_{i=1}^{m} h_i(Ax-b), \tag{25}$$

where $A \in^{m\times n}$, $b \in^{m}$, and

$$h_i(z) = \begin{cases} 0.5z_i^2, & \text{for } |z_i| \le 1 \\ |z_i| - 0.5, & \text{otherwise.} \end{cases}$$

The second class of problems is group LASSO:

$$\min_x \frac{1}{2}||Ax-b||_2^2 + \rho \sum_{i=1}^{N} ||x_i||_2 \tag{26}$$

where A and b are as above, x_i refers to the ith group of variables, with $i = 1, \ldots, N$, and $\rho > 0$ is a penalty parameter. Even though the second norm is nondifferentiable, the only potential issue is if all of the x values in a group approach 0. In practice, we can keep the group sizes large and easily find randomly generated problem instances where nondifferentiability is not encountered.

To generate random instances of these problems, we used the codes provided in Boyd et al. (2011b). For the Huber loss function, we ran 100 instances for four different problem sizes. These parameters are listed in $m = 5000$ and $n = 2000$, $m = 10000$ and $n = 2000$, $m = 100000$ and $n = 2000$, and $m = 100000$ and $n = 5000$. For group LASSO, we ran 100 instances for six problem sizes. The first four problem sizes are $m = 100000$ and $N = 2$, $m = 1500$ and $N = 4$, $m = 10000$ and $N = 4$, and $m = 100000$ and $N = 4$. The size of each block was a uniformly distributed random number between 1 and 1000, and n was calculated as the sum of the block sizes. The last two problem have the size $m = 100000$ and $N = 4$, but the random integer selected was between 1 and 2000 and 1 and 5000, respectively. Entries of the feature matrix A were generated as $N(0,1)$, and then the columns of A were normalized to have a unit l_2 norm. b was then generated using $b = Ax^{\text{true}} + v$ where $v \sim N(0, 0.1)$. ρ was chosen as $0.001\|A^T b\|_\infty$.

4.3. Do we need Powell restarts?

Of the 192 NLPs in the test set, 187 of them require at least one Powell restart. That is 97.4% - a significant portion. In fact, of the 5 problems that do not use Powell restarts, 4 conclude within the first two iterations without ever reaching a check for the Powell restart and 1 has an objective function that is the weighted sum of a quadratic term and a nonlinear term, where the weight and function values of the nonlinear term are negligible with respect to the quadratic term (and as such, it is effectively a QP). It is, therefore, safe to conclude that every NLP of interest requires at least one Powell restart and focus on these 187 problems in our ongoing analysis.

Of the 187 problems, our main code fails of 30 problems (8 exit due to linesearch failures, 3 exit due to function evaluation errors, and 19 reach the iteration limit of 10,000). The remaining 157 problems are reported as solved and show at least one Powell restart. For each of these problems, we calculated the percentage of Powell restart iterations as

$$\begin{aligned}\phi_i &= \text{Percentage of Powell restart iterations for Problem } i \\ &= \frac{\text{Number of iterations with Powell restarts for Problem } i}{\text{Total number of iterations for Problem } i}.\end{aligned}$$

The average value of ϕ is 45.5%, and its median is 45.7%. Note that we do not test for a Powell restart during an iteration that is already slated for a Beale restart, so the percentage of iterations that satisfy the Powell restart criterion (6) is actually slightly higher at around 55%.

Given the prevalence of Powell restarts, it may be natural to ask how the algorithm performs without them. When Powell restarts are disabled, the code still converges on the same number of problems (27 of the 30 failures remain the same, 3 get resolved, and there are 3 new failures). We get the same iteration count on 13 problems, the code performs better with Powell restarts on 101 problems (80% fewer iterations on average), and the code performs better without Powell restarts on 41 problems (30% fewer iterations on average). Therefore, there is a clear and significant advantage to Powell restarts, but the question remains as to whether or not we need a full restart every time the Powell restart criterion is satisfied.

4.4. The Impact of Cubic Regularization on the Powell Restart Criterion

We will now use an example to illustrate the impact of cubic regularization on the satisfaction of the Powell restart criterion (6). More specifically, we will examine how increasing values of λ impact the value of the fraction

$$\frac{|\nabla f(x_{k+1}(\lambda))^T \nabla f(x_k)|}{\|\nabla f(x_{k+1}(\lambda))\|^2}. \tag{27}$$

The example we have chosen is the problem *s206* (Bongartz et al. (1995)):

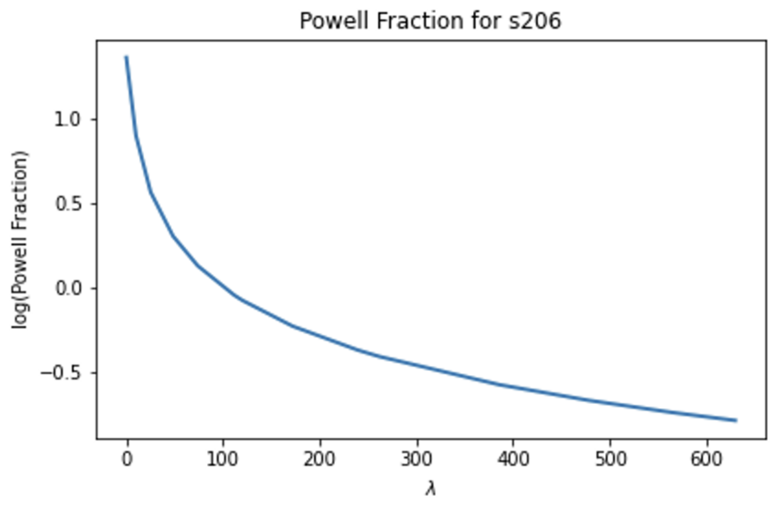
$$\min_{x_1,x_2} (x_2 - x_1^2)^2 + 100(1-x_1)^2.$$

Without cubic regularization, Conmin encounters a Powell restart in Iteration 3. The value of the Powell fraction (27) is 23.18, which is much larger than the threshold of 0.2. If we start to apply the cubic regularization with increasing values of λ for this iteration, we get the results shown in Figure 1.

Since the algorithm quickly increases the value of λ, it is hard to visualize a pattern of decrease with the values given in the table. As such, we have also evaluated the Powell fraction for more values of λ between 0 and 600 so that a smooth chart can be produced for Figure 1.

4.5. Numerical comparison

We start our numerical comparison with general problems of the form (1) from the CUTEr test set. Detailed results on these problems are provided in Tables 3-8 in the Appendix. The detailed results include the number of iterations, runtime in CPU seconds, and the objective function value at the reported solution for three algorithms. The first, labeled "With Powell Restarts," is the algorithm implemented in Conmin-CG and previously presented as Algorithm 1. The second, labeled "No Powell Restarts," is a modified version of Algorithm 1 with the check for Powell restarts removed, so that only Beale restarts are performed. This algorithm is included in the tables to support the results provided in Section 4.3. The last algorithm, labeled "Hybrid Cubic," implements Algorithm 2, which uses a hybrid approach where cubic regularization is only invoked when the Powell restart criterion (6) is satisfied.



λ	Powell Fraction (26)
0.00	23.18
115.90	0.87
120.44	0.84
239.17	0.43
478.35	0.22
629.78	0.16

Figure 1 λ **vs. the log of the Powell fraction in Iteration 3 of solving the problem s206. The first two iterations of the problem were solved without cubic regularization.**

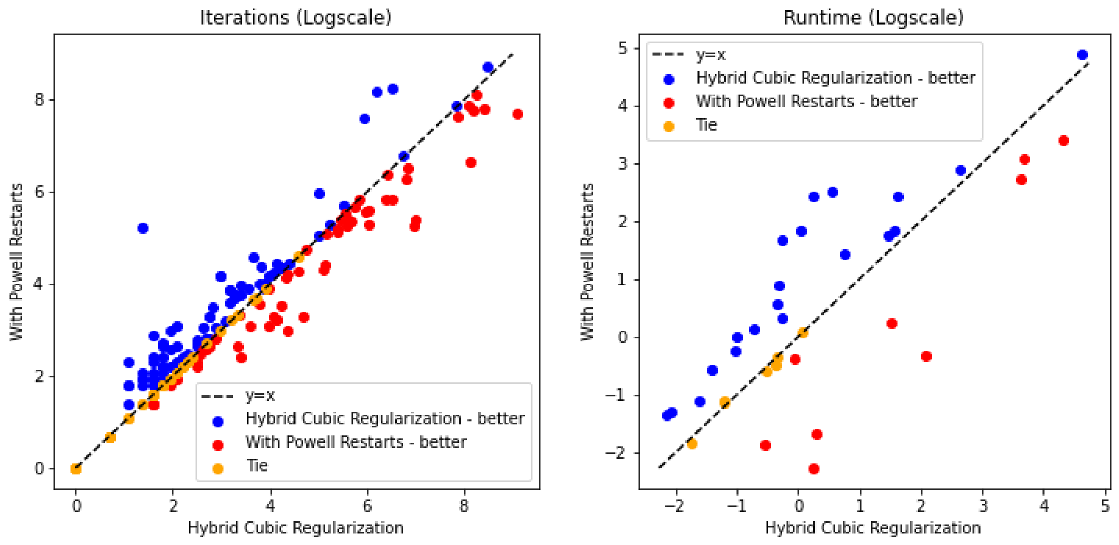


Figure 2 Pairwise comparisons of iterations and runtimes for Conmin-CG with Powell restarts and with hybrid cubic regularization. The iterations comparison was conducted on 180 out of the 230 unconstrained problems we solved from the CUTEr test set, and the runtimes comparison was conducted on 36 problems on which both solvers exhibited runtimes of at least 0.1 CPU seconds. Blue dots denote the problems where the code with hybrid cubic regularization outperforms the code with Powell restarts, red dots represent the opposite relationship, and orange dots represent a tie (or runtimes within 0.1 of each other). The dotted black line is $y=x$.

The results in Tables 3-8 show that the hybrid cubic regularization improves the number of iterations and the runtime on the CUTEr test set.

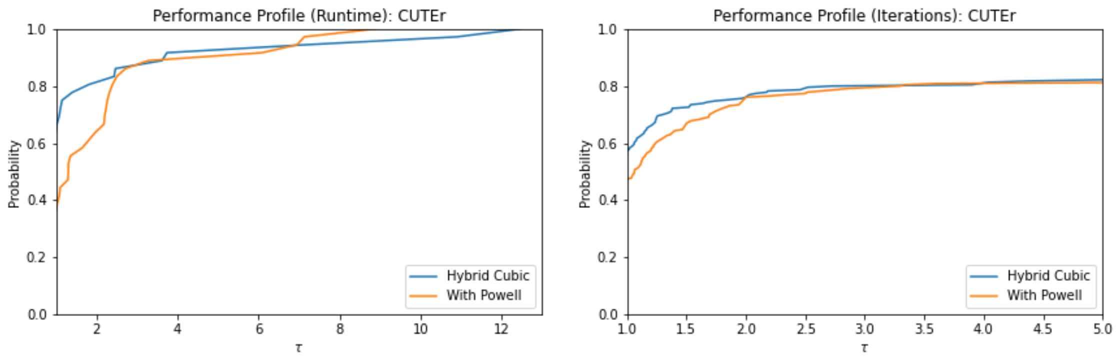


Figure 3 Performance profiles of the runtime and iteration results from CUTEr test set. The iterations comparison was conducted on 180 out of the 230 unconstrained problems we solved from the CUTEr test set, and the runtimes comparison was conducted on 36 problems on which both solvers exhibited runtimes of at least 0.1 CPU seconds.

- For overall success, we have that “With Powell Restarts” and “Hybrid Cubic” each solve 190 of the 230 problems, a rate of 82.6%. This includes 180 jointly solved problems, 10 problems solved by “With Powell Restarts” only, and 10 problems solved by “Hybrid Cubic” only.
- For iterations, we see in Figures 2 and 3 that “Hybrid Cubic” outperforms “With Powell Restarts” on the jointly solved problems. The graph on the left shows a scatterplot where each point represents one problem, with the coordinates equaling iteration counts by the two codes. (The line $y = x$ is included to help our assessment.) In this graph, there are 81 blue dots representing instances where “Hybrid Cubic” had fewer iterations, 59 red dots where “With Powell Restarts” had fewer iterations, and 40 orange dots where they had the same number of iterations. That means on 121 of the 180 jointly solved problems, or 67.2%, “Hybrid Cubic” exhibits the same or fewer number of iterations as “With Powell Restarts.”
- It is interesting to note that the improvement becomes slightly more pronounced if we use a typical iteration limit of 1,000 (instead of 10,000). In that case, there are 166 jointly solved problems. “Hybrid Cubic” performs fewer iterations on 76, “With Powell Restarts” performs fewer iterations on 50, and the two codes perform the same number of iterations on 40. That means on 116 of the 166 jointly solved problems, or 69.9%, “Hybrid Cubic” exhibits the same or fewer number of iterations as “With Powell Restarts.”
- There were 14 jointly solved problems where at least one solver performed over 1,000 iterations. On these instances, there is no clearly observed pattern, as such a high iteration typically indicates that the problem is severely ill-conditioned or that $f(x)$ is highly nonconvex. One example is the problem *watson*, with $n = 31$, and where

$$f(x) = \sum_{i=1}^{29} \left(\sum_{j=2}^{n} (j-1) \left(\frac{i}{29}\right)^{j-2} x_j - \left(\sum_{j=1}^{n} \left(\frac{i}{29}\right)^{j-2} x_j \right)^2 - 1 \right)^2 + x_1^2 + (x_2 - x_1^2 - 1)^2.$$

On this problem, “With Powell Restarts” performs 2248 and “Hybrid Cubic” performs 8919 iterations to reach a solution. Similar behavior is observed on *s371*, which is identical to *watson* but with $n = 9$. However, on *dixmaani-dixmaanl*, where $m = 1,000$, $n = 3,000$, and

$$f(x) = 1.0 + \sum_{i=1}^{n} \hat{a}\left(\frac{i}{n}\right)^2 x_i^2 + \sum_{i=1}^{n-1} \hat{b} x_i^2 (x_{i+1} + x_{i+1}^2)^2 + \sum_{i=1}^{2m} \hat{c} x_i^2 x_{i+m}^4 + \sum_{i=1}^{m} \hat{d}\left(\frac{i}{n}\right)^2 x_i x_{i+2m}$$

for different values of $\hat{a}$, $\hat{b}$, $\hat{c}$, and $\hat{d}$. On these problems, “With Powell Restarts” performs 2,000-6,000 iterations, which is 1,000-3,000 more than “Hybrid Cubic.”

- For the runtimes comparisons, we have taken a subset of the jointly solved problems, namely those that were solved in 0.1 or more CPU seconds by both solvers. (This is to ensure a fair comparison, as smaller runtimes can be easily influenced by other processes running on the same machine and/or exhibit very small differences.) The resulting set consisted of 36 problems, of which “With Powell Restarts” was faster on 13 and “Hybrid Cubic” was faster on 23. This means that “Hybrid Cubic” resulted in faster runtimes on 63.4% of the problems with runtimes of at least 0.1 CPU seconds.
- For runtime comparison, it may be a good idea to consider small differences as a tie. If runtimes within 0.1 CPU seconds of each other are considered a tie, then “With Powell Restarts” was faster on 9, “Hybrid Cubic” was faster on 20, and we considered 7 instances as a tie.

Next we will discuss the numerical results from randomly generated data on the Huber loss function (25) and group LASSO (26). In our testing, we compared three codes: Our implementations for “With Powell Restarts” and “Hybrid Cubic” in Matlab, and ADMM Boyd et al. (2011b).

- In Table 1, we see that the “Hybrid Cubic” had the shortest runtime out of the three, while ADMM took the longest to solve. In addition, “Hybrid Cubic” had less iterations than its non-regularized version, “With Powell Restarts”. We do not include ADMM iterations, given that iteration counts for ADMM and CGM have different interpretations, thus are incomparable.
- Results in Table 1 show that runtime and iteration count increase as problem sizes increase.
- Cubic regularization was invoked in every instance of Huber loss function problems.
- In Figure 4, we find that “With Powell Restarts” had a slight advantage over “Hybrid Cubic” in runtime, while ADMM was much slower.
- Cubic regularization was invoked on a small percentage of the problems in Figure 4.
- In group LASSO, when cubic regularization is invoked, it is typically on a single iteration and doesn’t have a big impact on the number of iterations. However, this implies that the additional computational burden of the cubic step is not offset by a reduction in the number of iterations. Therefore, in most instances where cubic is invoked, the runtime increases by a small amount.
- In Figure 4, “Hybrid Cubic” solves five problems that “With Powell Restarts’ cannot solve on its own. “With Powell Restarts’ solves two problems that ‘Hybrid Cubic” cannot solve. ADMM solves the most number of problems out of the three.

		ADMM	CGM - With Powell Restarts		CGM - Hybrid Cubic	
m	n	Time	Time	Iter	Time	Iter
5000	2000	0.94	0.60	26.48	0.56	13.43
10000	2000	1.60	0.72	16.09	0.54	7
100000	2000	12	3.04	7.43	2.46	4
100000	5000	47.58	11.63	10	9.12	5

Table 1 **Numerical results for randomly generated data on the Huber loss function** (25)**. m is the number of rows in the feature matrix** A**, n is the number of columns in the feature matrix** A**, Iter is the average iteration count and Time is the average run time in CPU seconds. Averages are computed over the 100 instances of each problem size.**

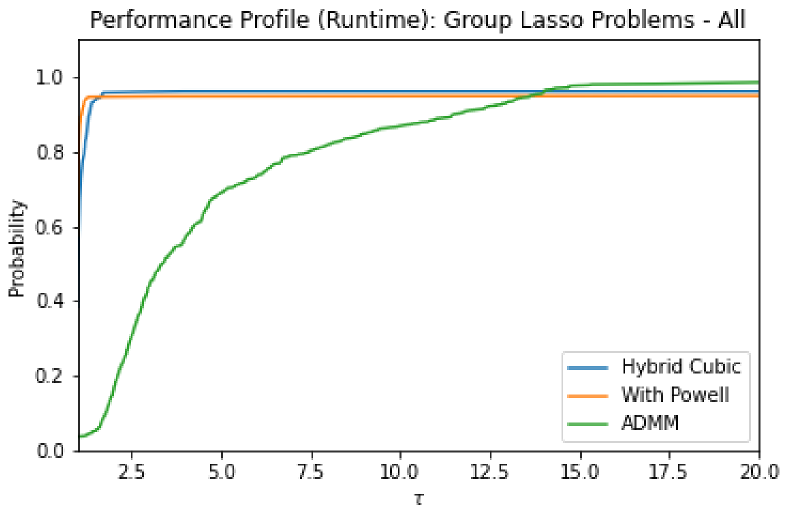


Figure 4 **Performance profile for runtime on randomly generated data on group LASSO** (26)**. Figure contains results from all 600 instances.**

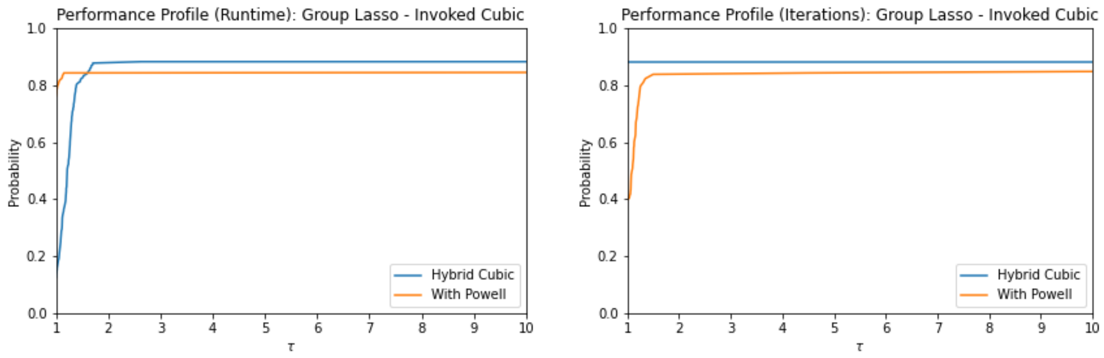


Figure 5 **Performance profile for runtime and iterations on randomly generated data on group LASSO** (26)**. Figure contains results from only the instances that invoked cubic regularization, 203 out of 600.**

5. Conclusion

Our goal in this paper was to incorporate cubic regularization selectively into a CGM framework so that the resulting approach would

- require fewer iteration counts in computational experiments than its non-regularized version,
- exhibit global convergence with fewer assumptions than its non-regularized version,
- require the same order of computational burden per iteration as its non-regularized version, and
- demonstrate faster overall runtime in computational experiments than its non-regularized version.

Our global convergence results in Section 3 and our numerical results in Section 4 showed that we were able to attain this goal fully.

We have some additional tasks to explore in future work. In our current implementation, we find the optimal α. However, we wish to explore how these results may be affected with a fixed step size. In addition, we also hope to apply our algorithm to real life datasets. We had used a few datasets from the UCI machine learning repository (Dua and Graff (2017)), but out of the several we chose, only two invoked cubic regularization. We hope to explore this in more detail and find additional applications for our algorithm.

References

Adeli H, Samant A (2000) An adaptive conjugate gradient neural network–wavelet model for traffic incident detection. *Computer-Aided Civil and Infrastructure Engineering* 15(4):251–260.

Azar AT (2013) Fast neural network learning algorithms for medical applications. *Neural Computing and Applications* 23(3):1019–1034.

Benson H, Shanno D (2012) Interior-point methods for nonconvex nonlinear programming: Cubic regularization. Technical report.

Benson H, Shanno D (2014) Interior-point methods for nonconvex nonlinear programming: Cubic regularization. *Computational Optimization and Applications* 58.

Benson HY, Shanno DF (2018) Cubic regularization in symmetric rank-1 quasi-newton methods. *Mathematical Programming Computation* 10(4):457–486.

Bongartz I, Conn AR, Gould N, Toint PL (1995) Cute: Constrained and unconstrained testing environment. *ACM Transactions on Mathematical Software (TOMS)* 21(1):123–160.

Boyd S, Parikh N, Chu E, Peleato B, Eckstein J (2011a) *Distributed optimization and statistical learning via the alternating direction method of multipliers* (Now Publishers Inc).

Boyd S, Parikh N, Chu E, Peleato B, Eckstein J (2011b) MATLAB scripts for alternating direction method of multipliers. URL `https://web.stanford.edu/~boyd/papers/admm/`.

Buhler C (2021) Conjugate gradient method of conmin in C and MATLAB. URL `https://cassiebuhler.github.io/software/`.

Cartis C, Gould N, Toint P (2011) Adaptive cubic regularisation methods for unconstrained optimization. Part I: motivation, convergence and numerical results. *Mathematical Programming, Series A* 127:245–295.

Çınar M, Engin M, Engin EZ, Ateşçi YZ (2009) Early prostate cancer diagnosis by using artificial neural networks and support vector machines. *Expert Systems with Applications* 36(3):6357–6361.

Crowder H, Wolfe P (1972) Linear convergence of the conjugate gradient method. *IBM Journal of Research and Development* 16(4):431–433.

Dua D, Graff C (2017) UCI machine learning repository. URL `http://archive.ics.uci.edu/ml`.

Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12(61):2121–2159, URL `http://jmlr.org/papers/v12/duchi11a.html`.

Fletcher R, Reeves CM (1964) Function minimization by conjugate gradients. *The computer journal* 7(2):149–154.

Fourer R, Gay D, Kernighan B (1993) *AMPL: A Modeling Language for Mathematical Programming* (Scientific Press).

Geoffrey Hinton NS, Swersky K (2012) Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, accessed: April 27, 2021.

Griewank A (1981) 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.

Hestenes MR, Stiefel E (1952) *Methods of conjugate gradients for solving linear systems*, volume 49 (NBS).

Khadse CB, Chaudhari MA, Borghate VB (2016a) Conjugate gradient back-propagation based artificial neural network for real time power quality assessment. *International Journal of Electrical Power & Energy Systems* 82:197–206.

Khadse CB, Chaudhari MA, Borghate VB (2016b) Electromagnetic compatibility estimator using scaled conjugate gradient backpropagation based artificial neural network. *IEEE Transactions on Industrial Informatics* 13(3):1036–1045.

Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Lenard M (1973) Practical convergence conditions for unconstrained optimization. *Mathematical Programming* 4:309–323.

Levenberg K (1944) A method for the solution of certain problems in least squares. *Quarterly of Applied Mathematics* 2:164–168.

Li T, Li Y, Liao M, Wang W, Zeng C (2016) A new wind power forecasting approach based on conjugated gradient neural network. *Mathematical Problems in Engineering* 2016.

Marquardt D (1963) An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics* 11:431–441.

MathWorks (2021) Choose a multilayer neural network training function- MATLAB deep learning toolbox documentation. `https://www.mathworks.com/help/deeplearning/ug/choose-a-multilayer-neural-network-training-function.html#responsive_offcanvas`, accessed: April 27, 2021.

Møller MF (1993) A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks* 6(4):525–533.

Nesterov Y, Polyak B (2006) Cubic regularization of Newton method and its global performance. *Mathematical Programming, Series A* 108:177–205.

Oren SS, Spedicato E (1976) Optimal conditioning of self-scaling variable metric algorithms. *Mathematical programming* 10(1):70–90.

Orozco J, García CAR (2003) Detecting pathologies from infant cry applying scaled conjugate gradient neural networks. *European Symposium on Artificial Neural Networks, Bruges (Belgium)*, volume 2003 (Citeseer).

Perry A (1978) Technical note—a modified conjugate gradient algorithm. *Operations Research* 26(6):1073–1078, URL `http://dx.doi.org/10.1287/opre.26.6.1073`.

Polak E, Ribiere G (1969) Note sur la convergence de methods de directions conjugres. *Revue Francaise Informat. Recherche Operationelle* 16:35–43.

Powell MJD (1976) Some convergence properties of the conjugate gradient method. *Mathematical Programming* 11(1):42–49.

Powell MJD (1977) Restart procedures for the conjugate gradient method. *Mathematical programming* 12(1):241–254.

Saini LM, Soni MK (2002) Artificial neural network-based peak load forecasting using conjugate gradient methods. *IEEE Transactions on Power Systems* 17(3):907–912.

Selvamuthu D, Kumar V, Mishra A (2019) Indian stock market prediction using artificial neural networks on tick data. *Financial Innovation* 5(1):1–12.

Shanno D (1978a) On the convergence of a new conjugate gradient algorithm. *SIAM Journal on Numerical Analylsis* 15(6):1247–1257.

Shanno DF (1978b) Conjugate gradient methods with inexact searches. *Mathematics of operations research* 3(3):244–256.

Shanno DF, Phua KH (1976) Algorithm 500: Minimization of unconstrained multivariate functions [e4]. *ACM Transactions on Mathematical Software (TOMS)* 2(1):87–94.

Sözen A, Arcaklioğlu E, Özalp M, Kanit EG (2004) Use of artificial neural networks for mapping of solar potential in turkey. *Applied Energy* 77(3):273–286.

Vanderbei R (1997) AMPL models. Http://orfe.princeton.edu/˜rvdb/ampl/nlmodels.

Weiser M, Deuflhard P, Erdmann B (2007) Affine conjugate adaptive Newton methods for nonlinear elastomechanics. *Optimization Methods and Software* 22(3):413–431.

Zeiler MD (2012) Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* .

6. Appendix

		With Powell Restarts			No Powell Restarts			Hybrid Cubic		
Name	n	Iter	Time	f(x*)	Iter	Time	f(x*)	Iter	Time	f(x*)
aircrftb	5	50	<0.1	3.1E-16	53	<0.1	3.8E-15	53	<0.1	5.2E-13
allinitu	4	20	<0.1	5.7E+00	9	<0.1	5.7E+00	7	<0.1	5.7E+00
arglina*	100	1	<0.1	1.0E+02	1	<0.1	1.0E+02	1	<0.1	1.0E+02
arglinb*	10	1	<0.1	4.6E+00	1	<0.1	4.6E+00	1	<0.1	4.6E+00
arglinc*	8	1	<0.1	6.1E+00	1	<0.1	6.1E+00	1	<0.1	6.1E+00
arwhead	5000	8	2.6E-01	-9.6E-10	9	<0.1	-1.4E-09	4	1.2E-01	-2.7E-09
bard	3	17	<0.1	8.2E-03	16	<0.1	8.2E-03	15	<0.1	8.2E-03
bdexp	5000	6	2.5E-01	8.0E-04	3	<0.1	7.3E-121	3	<0.1	7.3E-121
bdqrtic	1000	(E)			(E)			438	4.2E+00	4.0E+03
beale	2	11	<0.1	8.9E-18	11	<0.1	5.5E-19	10	<0.1	6.2E-21
biggs3	3	14	<0.1	1.7E-10	16	<0.1	4.5E-13	28	<0.1	1.1E-11
biggs5	5	84	<0.1	5.7E-03	90	<0.1	5.7E-03	169	<0.1	5.7E-03
biggs6	6	62	<0.1	3.4E-07	110	<0.1	7.2E-09	76	<0.1	3.7E-06
box2	2	5	<0.1	4.2E-15	7	<0.1	1.7E-14	5	<0.1	3.5E-14
box3	3	9	<0.1	4.4E-12	10	<0.1	2.4E-11	12	<0.1	2.9E-11
bratu1d	1001	(E)			(E)			(IL)		
brkmcc	2	4	<0.1	1.7E-01	5	<0.1	1.7E-01	5	<0.1	1.7E-01
brownal	10	7	<0.1	4.6E-16	7	<0.1	1.8E-15	5	<0.1	4.0E-15
brownbs	2	8	<0.1	2.1E-14	11	<0.1	2.4E-22	6	<0.1	2.2E-13
brownden	4	21	<0.1	8.6E+04	37	<0.1	8.6E+04	14	<0.1	8.6E+04
broydn7d	1000	339	3.3E-01	4.0E+02	332	2.0E-01	4.0E+02	345	2.0E-01	4.0E+02
brybnd	5000	12	3.3E-01	1.5E-12	14	3.0E-01	4.1E-12	13	3.0E-01	4.1E-13
chainwoo	1000	167	<0.1	1.0E+00	380	1.7E-01	4.6E+00	217	<0.1	1.0E+00
chnrosnb	50	218	<0.1	1.0E-13	258	<0.1	3.1E-14	232	<0.1	1.1E-13
cliff	2	(E)			(E)			(IL)		
clplatea	4970	877	5.9E+00	-1.3E-02	1306	8.4E+00	-1.3E-02	840	4.3E+00	-1.3E-02
clplateb	4970	538	6.2E+00	-7.0E+00	680	4.4E+00	-7.0E+00	900	4.8E+00	-7.0E+00
clplatec	4970	(IL)			(IL)			(IL)		
cosine	10000	7	7.9E-01	-1.0E+04	(E)			6	3.6E-01	-1.0E+04
cragglvy	5000	80	1.3E+00	1.7E+03	(E)			45	4.6E+00	1.7E+03
cube	2	14	<0.1	2.7E-16	17	<0.1	1.1E-17	16	<0.1	2.1E-20
curly10	10000	(IL)			(IL)			(IL)		
curly20	10000	(IL)			(IL)			(IL)		
curly30	10000	(E)			(E)			(IL)		
deconvu	51	269	<0.1	3.7E-10	321	<0.1	3.7E-10	413	<0.1	7.9E-08
denschna	2	7	<0.1	1.2E-19	7	<0.1	7.1E-15	7	<0.1	7.9E-15
denschnb	2	5	<0.1	1.2E-14	6	<0.1	9.7E-17	5	<0.1	2.3E-24
denschnc	2	10	<0.1	2.2E-17	11	<0.1	1.2E-16	8	<0.1	1.9E-19
denschnd	3	16	<0.1	3.7E-11	33	<0.1	7.8E-10	12	<0.1	2.4E-09
denschne	3	(E)			(E)			(IL)		

Table 2 **Numerical results on the unconstrained problems from the CUTEr test set (Vanderbei (1997)). Problem names that end in an asterisk are quadratic programming problems. n is the number of variables in the problem, Iter is the iteration count, Time is the run time in CPU seconds, and $f(x^*)$ is the objective value at the reported solution. (IL) and (E) denote that the solver reached its iteration limit and exited with an error, respectively.**

		With Powell Restarts			No Powell Restarts			Hybrid Cubic		
Name	n	Iter	Time	f(x*)	Iter	Time	f(x*)	Iter	Time	f(x*)
denschnf	2	6	<0.1	3.9E-21	8	<0.1	3.0E-16	4	<0.1	3.0E-16
dixmaana	3000	7	<0.1	1.0E+00	11	<0.1	1.0E+00	5	<0.1	1.0E+00
dixmaanb	3000	7	1.1E-01	1.0E+00	11	<0.1	1.0E+00	4	<0.1	1.0E+00
dixmaanc	3000	8	1.0E-01	1.0E+00	13	<0.1	1.0E+00	5	<0.1	1.0E+00
dixmaand	3000	10	1.7E-01	1.0E+00	16	<0.1	1.0E+00	5	<0.1	1.0E+00
dixmaane	3000	255	3.2E-01	1.0E+00	305	4.7E-01	1.0E+00	256	3.0E-01	1.0E+00
dixmaanf	3000	195	6.1E-01	1.0E+00	248	8.4E-01	1.0E+00	268	7.0E-01	1.0E+00
dixmaang	3000	227	7.0E-01	1.0E+00	239	8.5E-01	1.0E+00	268	7.2E-01	1.0E+00
dixmaanh	3000	181	5.6E-01	1.0E+00	316	1.0E+00	1.0E+00	220	6.0E-01	1.0E+00
dixmaani	3000	6084	1.1E+01	1.0E+00	4793	7.2E+00	1.0E+00	4806	5.0E+00	1.0E+00
dixmaanj	3000	3816	1.2E+01	1.0E+00	1409	4.6E+00	1.0E+00	675	1.7E+00	1.0E+00
dixmaank	3000	3597	1.1E+01	1.0E+00	663	2.3E+00	1.0E+00	499	1.3E+00	1.0E+00
dixmaanl	3000	2050	6.3E+00	1.0E+00	709	2.3E+00	1.0E+00	380	1.0E+00	1.0E+00
dixon3dq*	10	10	<0.1	2.2E-28	10	<0.1	2.2E-28	10	<0.1	2.4E-28
dqdrtic*	5000	6	<0.1	2.3E-15	6	<0.1	2.3E-15	6	<0.1	2.3E-15
dqrtic	5000	13	5.7E-01	1.0E-01	164	4.2E-01	9.5E-02	7	2.5E-01	9.5E-03
edensch	2000	17	<0.1	1.2E+04	17	<0.1	1.2E+04	16	<0.1	1.2E+04
eg2	1000	2	<0.1	-1.0E+03	2	<0.1	-1.0E+03	2	<0.1	-1.0E+03
engval1	5000	14	2.8E-01	5.5E+03	21	2.2E-01	5.5E+03	8	1.3E-01	5.5E+03
engval2	3	28	<0.1	5.9E-13	36	<0.1	9.0E-17	29	<0.1	1.5E-13
errinros	50	259	<0.1	4.0E+01	432	<0.1	4.0E+01	394	<0.1	4.0E+01
expfit	2	11	<0.1	2.4E-01	12	<0.1	2.4E-01	5	<0.1	2.4E-01
fletcbv2	100	97	<0.1	-5.1E-01	97	<0.1	-5.1E-01	97	<0.1	-5.1E-01
fletchcr	100	302	<0.1	3.5E-13	293	<0.1	1.3E-13	247	<0.1	2.4E-13
flosp2hl	650	(IL)			(IL)			(IL)		
flosp2hm	650	(IL)			(IL)			(IL)		
flosp2th	650	(IL)			(IL)			(IL)		
flosp2tl	650	(IL)			(IL)			(IL)		
flosp2tm	650	(IL)			(IL)			(IL)		
fminsrf2	1024	223	1.9E-01	1.0E+00	384	3.2E-01	1.0E+00	1106	1.4E+00	1.0E+00
fminsurf	1024	202	1.6E-01	1.0E+00	339	3.5E-01	1.0E+00	420	5.8E-01	1.0E+00
freuroth	5000	22	7.3E-01	6.1E+05	28	3.1E-01	6.1E+05	54	7.9E+00	6.1E+05
genhumps	5	36	<0.1	7.0E-14	34	<0.1	5.5E-13	24	<0.1	3.5E-12
genrose	500	2668	6.8E-01	1.0E+00	3682	5.5E-01	1.0E+00	2572	9.5E-01	1.0E+00
growth	3	196	<0.1	1.0E+00	159	<0.1	1.0E+00	(IL)		
growthls	3	169	<0.1	1.0E+00	177	<0.1	1.0E+00	(IL)		
gulf	3	41	<0.1	4.7E-13	32	<0.1	4.7E-09	41	<0.1	4.9E-10
hairy	2	18	<0.1	2.0E+01	11	<0.1	2.0E+01	5	<0.1	2.0E+01
hatfldd	3	22	<0.1	2.5E-07	26	<0.1	2.5E-07	36	<0.1	2.6E-07
hatflde	3	35	<0.1	4.4E-07	28	<0.1	4.4E-07	44	<0.1	4.4E-07
heart6ls	6	(IL)			(IL)			(IL)		
heart8ls	8	339	<0.1	1.5E-16	382	<0.1	1.5E-12	590	<0.1	5.2E-12
helix	3	21	<0.1	3.6E-17	25	<0.1	2.2E-17	18	<0.1	8.4E-23
hilberta*	10	8	<0.1	5.8E-10	8	<0.1	5.8E-10	8	<0.1	5.8E-10
hilbertb*	50	5	<0.1	2.1E-20	5	<0.1	2.1E-20	5	<0.1	2.1E-20

Table 3 Numerical results on the unconstrained problems from the CUTEr test set (Vanderbei (1997)). Problem names that end in an asterisk are quadratic programming problems. n is the number of variables in the problem, Iter is the iteration count, Time is the run time in CPU seconds, and $f(x^*)$ is the objective value at the reported solution. (IL) and (E) denote that the solver reached its iteration limit and exited with an error, respectively.

		With Powell Restarts			No Powell Restarts			Hybrid Cubic		
Name	n	Iter	Time	f(x*)	Iter	Time	f(x*)	Iter	Time	f(x*)
himmelbb	2	4	<0.1	1.7E-18	5	<0.1	1.7E-16	4	<0.1	5.8E-19
himmelbf	4	44	<0.1	3.2E+02	81	<0.1	3.2E+02	30	<0.1	3.2E+02
himmelbg	2	6	<0.1	1.6E-16	7	<0.1	9.4E-20	6	<0.1	1.3E-16
himmelbh	2	5	<0.1	-1.0E+00	5	<0.1	-1.0E+00	5	<0.1	-1.0E+00
humps	2	55	<0.1	2.1E-12	89	<0.1	4.6E-16	48	<0.1	2.2E-12
jensmp	2	15	<0.1	1.2E+02	16	<0.1	1.2E+02	6	<0.1	1.2E+02
kowosb	4	25	<0.1	3.1E-04	45	<0.1	3.1E-04	63	<0.1	3.1E-04
liarwhd	10000	14	1.8E+00	5.5E-15	18	3.3E-01	1.3E-17	12	7.1E-01	2.4E-19
loghairy	2	184	<0.1	1.8E-01	66	<0.1	1.8E-01	4	<0.1	6.2E+00
mancino	100	15	1.6E-01	3.6E-15	15	1.9E-01	5.5E-15	15	1.8E-01	3.9E-15
maratosb	2	(E)			(E)			4	<0.1	-1.0E+00
methanb8	31	2101	<0.1	4.9E-05	1377	<0.1	5.2E-05	2602	1.5E-01	6.9E-05
methanl8	31	(IL)			(IL)			(IL)		
mexhat	2	9	<0.1	-4.0E-02	6	<0.1	-4.0E-02	6	<0.1	-4.0E-02
meyer3	3	(E)			(E)			(IL)		
minsurf	36	13	<0.1	1.0E+00	17	<0.1	1.0E+00	15	<0.1	1.0E+00
msqrtals	1024	3376	2.2E+01	2.7E-08	3658	2.9E+01	1.1E-08	3866	4.0E+01	4.7E-08
msqrtbls	1024	2376	1.6E+01	3.4E-09	2741	2.1E+01	1.7E-08	3630	3.8E+01	6.9E-09
nasty*	2	(E)			(E)			(E)		
ncb20	1010	(E)			(E)			110	6.9E+00	1.7E+03
ncb20b	1000	(E)			(E)			34	7.4E+00	1.7E+03
nlmsurf	15129	2467	1.3E+02	3.9E+01	3857	9.9E+01	3.9E+01	4582	1.0E+02	3.9E+01
noncvxu2	1000	2684	1.1E+00	2.3E+03	2814	9.3E-01	2.3E+03	3325	1.1E+00	2.3E+03
noncvxun	1000	22	<0.1	2.3E+03	20	<0.1	2.3E+03	8	<0.1	2.3E+03
nondia	9999	9	1.2E+00	2.8E-15	8	2.8E-01	3.9E-24	6	4.8E-01	3.2E-12
nondquar	10000	348	1.8E+01	6.4E-05	1523	8.8E+00	6.8E-05	683	1.4E+01	9.5E-05
nonmsqrt	9	670	<0.1	7.5E-01	282	<0.1	7.5E-01	925	<0.1	7.5E-01
osbornea	5	258	<0.1	5.5E-05	289	<0.1	5.5E-05	(IL)		
osborneb	11	162	<0.1	4.0E-02	143	<0.1	4.0E-02	175	<0.1	4.0E-02
palmer1c*	8	(IL)			(E)			(IL)		
palmer1d*	7	(E)			(E)			8357	5.0E-01	6.5E-01
palmer1e	8	(IL)			(IL)			4	<0.1	0.0E+00
palmer2c*	8	(IL)			(IL)			(IL)		
palmer2e	8	(IL)			(IL)			(IL)		
palmer3c*	8	5844	1.0E-01	2.0E-02	(IL)			(IL)		
palmer3e	8	(IL)			9272	1.9E-01	5.1E-05	(IL)		
palmer4c*	8	3319	<0.1	5.0E-02	4021	<0.1	5.1E-02	(IL)		
palmer4e	8	4453	<0.1	1.5E-04	6286	1.3E-01	1.5E-04	(IL)		
palmer5c*	6	6	<0.1	2.1E+00	6	<0.1	2.1E+00	6	<0.1	2.1E+00
palmer5d*	4	9	<0.1	8.7E+01	9	<0.1	8.7E+01	8	<0.1	8.7E+01
palmer6c*	8	2336	<0.1	2.1E-02	6634	1.0E-01	1.9E-02	(IL)		
palmer7c*	8	8231	1.4E-01	6.2E-01	(IL)			(IL)		
palmer8c*	8	1863	<0.1	1.6E-01	4321	<0.1	1.7E-01	(IL)		

Table 4 Numerical results on the unconstrained problems from the CUTEr test set (Vanderbei (1997)). Problem names that end in an asterisk are quadratic programming problems. n is the number of variables in the problem, Iter is the iteration count, Time is the run time in CPU seconds, and $f(x^*)$ is the objective value at the reported solution. (IL) and (E) denote that the solver reached its iteration limit and exited with an error, respectively.

		With Powell Restarts			No Powell Restarts			Hybrid Cubic		
Name	n	Iter	Time	f(x*)	Iter	Time	f(x*)	Iter	Time	f(x*)
penalty1	1000	25	<0.1	9.7E-03	49	<0.1	9.7E-03	25	<0.1	9.7E-03
penalty2	100	86	<0.1	9.7E+04	(E)			63	<0.1	9.7E+04
penalty3	100	(E)			(E)			(IL)		
pfit1	3	40	<0.1	2.9E-04	38	<0.1	2.9E-04	26	<0.1	2.9E-04
pfit1ls	3	40	<0.1	2.9E-04	38	<0.1	2.9E-04	26	<0.1	2.9E-04
pfit2	3	44	<0.1	1.2E-02	50	<0.1	1.2E-02	26	<0.1	1.2E-02
pfit2ls	3	44	<0.1	1.2E-02	50	<0.1	1.2E-02	26	<0.1	1.2E-02
pfit3	3	66	<0.1	8.2E-02	56	<0.1	8.2E-02	20	<0.1	8.2E-02
pfit3ls	3	66	<0.1	8.2E-02	56	<0.1	8.2E-02	20	<0.1	8.2E-02
pfit4	3	(E)			62	<0.1	2.6E-01	19	<0.1	2.6E-01
pfit4ls	3	(E)			62	<0.1	2.6E-01	19	<0.1	2.6E-01
powellsg	4	78	<0.1	2.0E-10	67	<0.1	2.2E-11	70	<0.1	5.6E-10
power*	1000	(IL)			(IL)			(IL)		
quartc	10000	14	2.4E+00	1.6E-01	150	9.9E-01	5.0E-01	6	7.4E-01	7.3E-01
rosenbr	2	27	<0.1	9.4E-18	24	<0.1	7.1E-17	16	<0.1	1.1E-16
s201*	2	2	<0.1	4.7E-27	2	<0.1	4.7E-27	2	<0.1	4.7E-27
s202	2	9	<0.1	4.9E+01	8	<0.1	4.9E+01	7	<0.1	4.9E+01
s204	2	5	<0.1	1.8E-01	5	<0.1	1.8E-01	5	<0.1	1.8E-01
s205	2	9	<0.1	1.1E-16	10	<0.1	1.0E-17	8	<0.1	5.0E-13
s206	2	4	<0.1	1.9E-16	5	<0.1	2.1E-19	5	<0.1	1.9E-24
s207	2	7	<0.1	2.4E-13	8	<0.1	4.3E-14	8	<0.1	2.2E-13
s208	2	27	<0.1	9.4E-18	24	<0.1	7.1E-17	16	<0.1	1.1E-16
s209	2	98	<0.1	1.2E-18	86	<0.1	8.1E-22	39	<0.1	1.8E-19
s210	2	389	<0.1	5.3E-21	372	<0.1	3.6E-20	148	<0.1	1.7E-18
s211	2	14	<0.1	2.7E-16	17	<0.1	1.1E-17	16	<0.1	2.1E-20
s212	2	9	<0.1	6.9E-22	12	<0.1	1.1E-25	8	<0.1	2.2E-15
s213	2	10	<0.1	1.6E-12	14	<0.1	1.1E-09	12	<0.1	1.1E-09
s240*	3	2	<0.1	3.8E-15	2	<0.1	3.8E-15	2	<0.1	3.8E-15
s243	3	9	<0.1	8.0E-01	9	<0.1	8.0E-01	9	<0.1	8.0E-01
s245	3	11	<0.1	1.7E-15	11	<0.1	2.7E-17	30	<0.1	6.1E-14
s246	3	17	<0.1	1.8E-16	18	<0.1	5.5E-18	18	<0.1	4.1E-20
s256	4	78	<0.1	2.0E-10	67	<0.1	2.2E-11	70	<0.1	5.6E-10
s258	4	48	<0.1	6.2E-13	27	<0.1	1.8E-15	24	<0.1	3.1E-17
s260	4	48	<0.1	6.2E-13	27	<0.1	1.8E-15	24	<0.1	3.1E-17
s261	4	27	<0.1	1.2E-09	37	<0.1	6.4E-10	59	<0.1	1.1E-09
s266	5	12	<0.1	1.0E+00	13	<0.1	1.0E+00	10	<0.1	1.0E+00
s267	5	75	<0.1	2.6E-03	68	<0.1	7.7E-09	163	<0.1	5.5E-07
s271*	6	6	<0.1	0.0E+00	6	<0.1	0.0E+00	6	<0.1	0.0E+00
s272	6	34	<0.1	5.7E-03	61	<0.1	5.7E-03	69	<0.1	5.7E-03
s272a	6	67	<0.1	3.4E-02	68	<0.1	3.4E-02	(IL)		
s273	6	11	<0.1	5.3E-18	14	<0.1	6.3E-15	6	<0.1	1.0E-14
s274*	2	2	<0.1	2.6E-24	2	<0.1	2.6E-24	2	<0.1	2.6E-24

Table 5 Numerical results on the unconstrained problems from the CUTEr test set (Vanderbei (1997)). Problem names that end in an asterisk are quadratic programming problems. n is the number of variables in the problem, Iter is the iteration count, Time is the run time in CPU seconds, and $f(x^*)$ is the objective value at the reported solution. (IL) and (E) denote that the solver reached its iteration limit and exited with an error, respectively.

		With Powell Restarts			No Powell Restarts			Hybrid Cubic		
Name	n	Iter	Time	f(x*)	Iter	Time	f(x*)	Iter	Time	f(x*)
s275*	4	3	<0.1	6.0E-12	3	<0.1	6.0E-12	3	<0.1	6.0E-12
s276*	6	3	<0.1	1.5E-12	3	<0.1	1.5E-12	3	<0.1	1.5E-12
s281a*	10	11	<0.1	2.0E-15	11	<0.1	2.0E-15	11	<0.1	1.3E-16
s282	10	212	<0.1	2.7E-15	220	<0.1	1.2E-16	292	<0.1	1.3E-15
s283	10	52	<0.1	1.5E-09	117	<0.1	7.3E-09	49	<0.1	2.4E-09
s286	20	24	<0.1	6.3E-16	27	<0.1	7.2E-14	22	<0.1	1.7E-17
s287	20	54	<0.1	2.4E-17	36	<0.1	9.3E-16	30	<0.1	9.4E-15
s288	20	70	<0.1	3.2E-10	80	<0.1	4.0E-10	58	<0.1	6.9E-10
s289	30	4	<0.1	0.0E+00	4	<0.1	0.0E+00	3	<0.1	0.0E+00
s290*	2	2	<0.1	1.1E-31	2	<0.1	1.1E-31	2	<0.1	3.1E-32
s291*	10	10	<0.1	2.8E-33	10	<0.1	2.8E-33	10	<0.1	5.5E-33
s292*	30	28	<0.1	7.1E-15	28	<0.1	7.1E-15	28	<0.1	7.1E-15
s293*	50	39	<0.1	3.0E-15	39	<0.1	3.0E-15	39	<0.1	3.0E-15
s294	6	50	<0.1	2.2E-15	54	<0.1	2.2E-17	48	<0.1	1.2E-20
s295	10	86	<0.1	1.2E-15	90	<0.1	3.8E-18	81	<0.1	2.0E-14
s296	16	115	<0.1	5.7E-14	134	<0.1	7.6E-15	117	<0.1	1.4E-14
s297	30	203	<0.1	1.6E-13	280	<0.1	1.3E-14	190	<0.1	1.1E-14
s298	50	288	<0.1	1.2E-14	413	<0.1	7.6E-15	317	<0.1	1.7E-14
s299	100	590	<0.1	5.3E-14	809	<0.1	1.1E-14	618	<0.1	3.7E-14
s300*	20	20	<0.1	-2.0E+01	20	<0.1	-2.0E+01	20	<0.1	-2.0E+01
s301*	50	50	<0.1	-5.0E+01	50	<0.1	-5.0E+01	50	<0.1	-5.0E+01
s302*	100	100	<0.1	-1.0E+02	100	<0.1	-1.0E+02	100	<0.1	-1.0E+02
s303	20	15	<0.1	6.7E-30	18	<0.1	3.3E-19	12	<0.1	8.1E-16
s304	50	11	<0.1	7.0E-14	19	<0.1	6.6E-25	9	<0.1	3.3E-24
s305	100	13	<0.1	4.2E-23	30	<0.1	1.6E-15	14	<0.1	3.1E-28
s308	2	7	<0.1	7.7E-01	8	<0.1	7.7E-01	7	<0.1	7.7E-01
s309	2	6	<0.1	2.9E-01	7	<0.1	2.9E-01	7	<0.1	2.9E-01
s311	2	6	<0.1	1.7E-14	7	<0.1	2.1E-23	5	<0.1	1.1E-19
s312	2	33	<0.1	5.9E+00	26	<0.1	5.9E+00	17	<0.1	5.9E+00
s314	2	4	<0.1	1.7E-01	5	<0.1	1.7E-01	5	<0.1	1.7E-01
s333	3	(E)			(E)			3	<0.1	0.0E+00
s334	3	17	<0.1	8.2E-03	16	<0.1	8.2E-03	15	<0.1	8.2E-03
s350	4	25	<0.1	3.1E-04	45	<0.1	3.1E-04	63	<0.1	3.1E-04
s351	4	65	<0.1	3.2E+02	46	<0.1	3.2E+02	54	<0.1	3.2E+02
s352*	4	4	<0.1	9.0E+02	4	<0.1	9.0E+02	4	<0.1	9.0E+02
s370	6	72	<0.1	2.3E-03	80	<0.1	2.3E-03	98	<0.1	2.3E-03
s371	9	771	<0.1	1.8E-06	1838	<0.1	4.0E-06	3361	1.3E-01	5.2E-06
s379	11	159	<0.1	4.0E-02	159	<0.1	4.0E-02	151	<0.1	4.0E-02
s386*	2	2	<0.1	4.7E-27	2	<0.1	4.7E-27	2	<0.1	4.7E-27
sbrybnd	5000	(IL)			(IL)			(IL)		
schmvett	10000	(E)			(E)			(E)		
scosine	10000	(IL)			(IL)			(IL)		

Table 6 Numerical results on the unconstrained problems from the CUTEr test set (Vanderbei (1997)). Problem names that end in an asterisk are quadratic programming problems. n is the number of variables in the problem, Iter is the iteration count, Time is the run time in CPU seconds, and $f(x^*)$ is the objective value at the reported solution. (IL) and (E) denote that the solver reached its iteration limit and exited with an error, respectively.

		With Powell Restarts			No Powell Restarts			Hybrid Cubic		
Name	n	Iter	Time	f(x*)	Iter	Time	f(x*)	Iter	Time	f(x*)
scurly10	10000	(IL)			(IL)			(IL)		
scurly20	10000	(IL)			(IL)			(IL)		
scurly30	10000	(IL)			(IL)			(IL)		
sineval	2	49	<0.1	2.8E-23	50	<0.1	4.3E-22	35	<0.1	5.2E-23
sinquad	10000	194	3.0E+01	4.1E-11	3259	4.0E+01	8.0E-09	1049	7.5E+01	2.6E-05
sisser	2	7	<0.1	2.9E-10	12	<0.1	7.7E-11	4	<0.1	2.7E-10
snail	2	20	<0.1	1.7E-14	69	<0.1	3.1E-23	78	<0.1	2.5E-22
srosenbr	10000	27	4.2E+00	6.7E-15	27	3.4E-01	3.3E-16	109	2.1E+00	5.0E-12
testquad*	1000	(IL)			(IL)			(IL)		
tointgss	10000	6	1.0E+00	1.0E+01	4	2.3E-01	1.0E+01	3	3.7E-01	1.0E+01
tquartic	10000	9	1.4E+00	7.1E-15	10	2.8E-01	4.2E-14	9	7.6E-01	1.8E-12
tridia*	10000	(IL)			4128	1.1E+01	4.8E-15	5118	1.5E+01	4.9E-15
vardim	100	10	<0.1	4.6E-17	21	<0.1	7.5E-18	3	<0.1	7.3E-26
vibrbeam	8	(IL)			(IL)			(IL)		
watson	31	2248	1.0E-01	1.1E-08	4969	2.6E-01	9.5E-09	8919	1.3E+00	9.7E-09
woods	10000	55	5.3E+00	1.8E-15	104	8.9E-01	6.5E-10	44	7.7E-01	6.2E-11
yfitu	3	67	<0.1	6.7E-13	63	<0.1	6.8E-13	78	<0.1	4.1E-06
zangwil2*	2	1	<0.1	-1.8E+01	1	<0.1	-1.8E+01	1	<0.1	-1.8E+01

Table 7 Numerical results on the unconstrained problems from the CUTEr test set (Vanderbei (1997)). Problem names that end in an asterisk are quadratic programming problems. n is the number of variables in the problem, Iter is the iteration count, Time is the run time in CPU seconds, and $f(x^*)$ is the objective value at the reported solution. (IL) and (E) denote that the solver reached its iteration limit and exited with an error, respectively.