Implementing the cubic and adaptive cubic regularization algorithms

Corinne Jones

June 8, 2016

1 Introduction

Recent work by Nesterov and Polyak (2006) and Cartis et al. (2011) shows that with slight modifications to the well-known Newton's method, global convergence to second-order critical points of an unconstrained optimization problem can be achieved under certain conditions. The main necessary condition is that the Hessian be Lipschitz continuous. This convergence guarantee is quite appealing, as we fairly frequently encounter problems with many saddle points (e.g., in some formulations of the matrix completion problem) where we want to be able to ensure that we converge to a minimum. Other recent work on this issue includes that by Anandkumar and Ge (2016), who use cubic regularization as part of their algorithm that converges to third order local optima.

No open source code exists online for either the cubic or adaptive cubic regularization algorithms. In addition, Nesterov and Polyak (2006) note at the end of their paper that "Indeed, it could be too ambitious to derive from our purely theoretical results any conclusion on the practical efficiency of corresponding algorithmic implementations". While Cartis et al. (2011) do perform some experiments, they do not use an estimate of the Hessian, which they tout as one of the benefits of adaptive cubic regularization, nor do they compare their results to those of other commonly used algorithms.

The goal of this project is to provide an open-source version of both cubic and adaptive cubic regularization, to test these algorithms on a variety of problems, and compare them with other commonly used algorithms. To that end, I first describe the cubic and adaptive cubic regularization algorithms in the next section. Following this, I present and discuss results from running the algorithms on a variety of problems. The code for the implementations may be found on GitHub at `https://github.com/cjones6/cubic_reg`.

2 Background

In this section I will briefly describe the cubic and adaptive cubic regularization algorithms. Following this I will provide details as to how to solve the cubic subproblem.

2.1 Overview of algorithms

The well-known algorithm Newton's method minimizes a second-order approximation to a function $f : \mathbb{R}^n \to \mathbb{R}$ at each iteration. More specifically, at each iteration, if the algorithm is currently at point x_k then it next moves to the point

$$x_{k+1} = \text{argmin}_y \left[f(x_k) + \langle f'(x_k), y - x_k \rangle + \frac{1}{2} \langle f''(x_k)(y - x_k), y - x_k \rangle \right],$$

i.e., the update rule is $x_{k+1} = x_k - [f''(x_k)]^{-1} f'(x_k)$. The downsides of this algorithm are that the Hessian will not always be invertible, and that the algorithm could converge to a saddle point or a local maximum. While an abundance of work exists that evades the invertibility problem, Nesterov and Polyak (2006) provide one of the first analyses of the global convergence of an algorithm related to Newton's method.

The idea behind the cubic regularization algorithm of Nesterov and Polyak (2006) is that we can upper bound our function f by a cubic function, and minimize this at every step. Specifically, assuming the Hessian of our function f is Lipschitz continuous with constant L, we can set

$$x_{k+1} = \text{argmin}_y \left[f(x_k) + \langle f'(x_k), y - x_k \rangle + \frac{1}{2} \langle f''(x_k)(y - x_k), y - x_k \rangle + \frac{M}{6} \|y - x\|^3 \right],$$

where M is either the Lipschitz constant or an approximation of it found via line search. The line search finds a value of M such that the function value at the next point will be below the the value at the minimizer of the current cubic approximation. Solving this cubic subproblem at each step guarantees that the objective function value will decrease unless the algorithm is at a minimum or possibly a degenerate saddle point.

Similarly to cubic regularization, adaptive cubic regularization also solves a third order approximation to the function f at each iteration. Letting B_k be a symmetric approximation to the Hessian at x_k, Cartis et al. (2011) use the model

$$m_k(y) = f(x_k) + \langle f'(x_k), y - x_k \rangle + \frac{1}{2} \langle B(x_k)(y - x_k), y - x_k \rangle + \frac{1}{3} \sigma_k \|y - x\|^3,$$

where σ_k is a line search parameter updated at every iteration depending on how well the proposed update (the minimizer of $m_k(y)$) does. If the minimizer y_k^* of $m_k(y)$ is such that the ratio $\rho_k = \frac{f(x_k) - f(y_k^*)}{f(x_k) - m_k(y_k^*)}$ is larger than some constant, then we will accept the update, move to y_k^*, and decrease the value of σ_k. Intuitively, this says that we move if the amount we would decrease is a significant fraction of the expected decrease based on the approximation m_k. If this fraction is not large enough we stay where we are and increase the value of σ_k until we find a value y_k^* that makes ρ_k large enough.

2.2 Solving the cubic subproblem

Both the cubic and adaptive cubic regularization algorithms require solving a cubic subproblem at each iteration. At first glance this task appears difficult, as the problem is non-convex. However, it turns out that for any $M > 0$ this subproblem is equivalent to the convex problem

$$\sup_{r \in \mathcal{D}} \left[-\frac{1}{2} \langle \left(H + \frac{Mr}{2} I \right)^{-1} g, g \rangle - \frac{M}{12} r^3 \right],$$

where $\mathcal{D} = \{r \in \mathbb{R} : H + \frac{M}{2} rI \succ 0, r \geq 0\}$ and g and H are the gradient and Hessian, respectively, at the current point.

This new problem is similar to a commonly-encountered problem in trust region methods discussed in Conn et al. (2000). An interior solution to our problem satisfies

$$r = \| \left(H + \frac{Mr}{2} I \right)^{-1} g \|, \quad r \geq \frac{2}{M} (-\lambda_n(H))_+, \tag{1}$$

where $\lambda_n(H)$ is the smallest eigenvalue of H. In contrast, the trust region problems have a constant left-hand side of the equation on the left. Note that we can rewrite $r = \| \left(H + \frac{Mr}{2} I \right)^{-1} g \|$ as

$$r^2 = \sum_{i=1}^{N} \frac{\tilde{g}_i^2}{(\lambda_i + \frac{M}{2} r)^2} \equiv \psi(r),$$

where λ_i is the ith largest eigenvalue of H and $\tilde{g}_i$ is ith element of Hg. Hence, the right hand side of this equation has poles at $-\lambda_i$ for each i unless $\tilde{g}_i = 0$. Moreover, the right hand side is decreasing to zero as r increases when $\frac{M}{2} r > -\lambda_i$. Therefore, there are a couple of possible cases to consider for the cubic subproblem.

The hard case occurs when $\tilde{g}_n = 0$ and $\lambda_n = 0$. In this case, we know that one solution is the limiting value of $-\left(H + \frac{Mr}{2} I \right)^+ g$ as $r \to \frac{2}{M} \lambda_n$, which I will denote by s_{cri} as is done in Conn et al. (2000). However, $H + \frac{Mr}{2} I$ is not invertible, so there are also other solutions, which are of the form $(H + \frac{M\lambda_n}{2} I)(s_{cri} + \alpha u_n) = -g$ for some constant α, where u_n is the eigenvector corresponding to λ_n. To choose the correct α, we find the one satisfying $\|s_{cri} + \alpha u_n\|_2 = -\frac{2}{M} \lambda_1$. In all other cases $\psi(r) \to \infty$ as $r \to \max(-\lambda_n, 0)$ and approaches zero as $r \to \infty$. Hence, there is a unique solution to the equation $r^2 = \psi(r)$ subject to $r \geq \frac{2}{M} (-\lambda_n(H))_+$.

To solve the problem in equation 1 in the nice case, we can use Newton-Raphson with some added safeguards on the equation

$$\phi(\lambda) \equiv \frac{1}{\|s(\lambda)\|_2} - \frac{M}{2\lambda} = 0, \tag{2}$$

where $\lambda = \frac{M}{2} r$ and $s(\lambda) = (H + \lambda I)^{-1} g$. Recall that the Newton-Raphson updates are $\lambda^+ = \lambda - \frac{\phi(\lambda)}{\phi'(\lambda)}$. In our case we have $\phi'(\lambda) = -\frac{\langle s(\lambda), \nabla_\lambda s(\lambda) \rangle}{\|s(\lambda)\|_2^3} + \frac{M}{2\lambda^2}$ where $\nabla_\lambda s(\lambda) = -H(\lambda)^{-1} s(\lambda)$. The analysis of the trust region subproblem in Conn et al. (2000) largely carries over to this problem, with the exception of the convergence criteria. In particular, we can apply Algorithm 7.3.6 from Conn et al. (2000) after modifying the Newton-Raphson updates and adding a small value to λ if $\lambda = 0$ at the beginning.

Algorithm 7.3.6 from Conn et al. (2000) requires finding the minimum eigenvalue of H so that we can start in a region where Newton-Raphson will converge to the unique solution of our problem. An alternative to this is Algorithm 7.3.4 from Conn et al. (2000), which does not require this and instead iteratively updates bounds on the region of convergence. However, I did not implement this because it is rather involved and I did not have time.

3 Results

I implemented both the cubic and adaptive cubic regularization algorithms in Python and the code may be found on GitHub at `https://github.com/cjones6/cubic_reg`. The main decision I had to make was which algorithm to use to solve the cubic subproblem, and for this I adapted Algorithm 7.3.6 from Conn et al. (2000). The default parameter settings and line search methods mostly come from the suggestions in Nesterov and Polyak (2006) and Cartis et al. (2011). However, Nesterov and Polyak (2006) do not specify how to set L_0, the initial value for M when L is unknown. As we only need a lower bound on this value, I set it to be an estimate of L at the initial point. I also allow the user to choose the Hessian update method when running adaptive cubic regularization; the choices are "exact", "rank_one" (the symmetric rank one update), and "broyden" (the Powell-symmetric-Broyden update).

The next subsection describes the initial testing of the algorithms and demonstrates that they work. Following that I display results from running the algorithms on more complicated functions from the CUTEr test set of Gould et al. (2003).

3.1 Initial testing

Initial unit testing provided promising results, with the cubic regularization algorithm converging to the minimum of convex function $f(x, y) = x^2 + y^2$ in one step when starting at (2,2). Adaptive cubic regularization took five steps before converging. For both of these algorithms I used a convergence tolerance on the gradient of 10^{-4}.

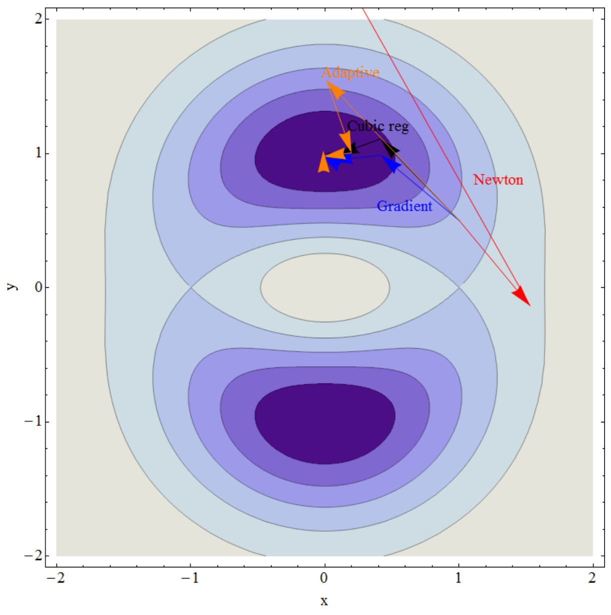
Next, I ran the algorithms on a more complicated and non-convex function, $f(x, y) = -(x^2 + 3y^2)e^{1-x^2-y^2}$. Figure 1a displays the paths that cubic regularization, adaptive cubic regularization, gradient descent, and Newton's algorithm take. Newton's method fails to converge to one of the local minima, while both cubic and adaptive cubic regularization do converge. Figure 1b, a quiver plot, displays the next step the cubic regularization algorithm will take based on where it currently is. The points (0,1) and (0, -1) are the local minima.

One additional necessary test was whether the algorithm performed properly when it reached an instance of the hard case of the cubic subproblem. Nesterov and Polyak (2006) describe one such instance in their Example 4 on page 200. For this problem, which is included in the unit tests code, the actual solution is $(1, \sqrt{3})$ and the code returns the correct answer up to 15 decimal places!

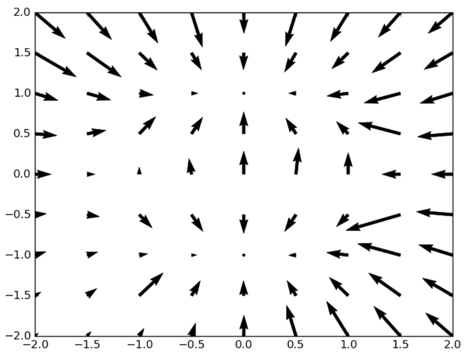
3.2 Results on subset of the CUTEr test set

In addition to running cubic and adaptive cubic regularization on the functions discussed above, I also ran the algorithms on a subset of the functions using a Python interface[1] to the

[1] http://fides.fe.uni-lj.si/ arpadb/software-pycuter.html



(a) Solution path of algorithms



(b) Direction of next location of cubic regularization when starting at various points

Figure 1: Results from minimizing the function $f(x) = -(x^2 + 3y^2)e^{1-x^2-y^2}$

CUTEr test set of Gould et al. (2003). These functions and their starting points are listed in the Appendix (located in a separate file) in Tables 6 and 7.

For sake of comparison with Cartis et al. (2011), I set the dimension of the problems on which I tested the algorithms to be 100 and used the same parameter values and convergence criterion (norm of the gradient less than 10^{-5}). The results I report for adaptive cubic regularization are from running it using Powell-symmetric-Broyden updates, as using an approximation to the Hessian is supposedly one of the main benefits of adaptive cubic regularization. This is one main difference between my implementation and that of Cartis et al. (2011). The other main difference is that my convergence criterion for the cubic subproblem is slightly different.

The convergence results for cubic regularization, adaptive cubic regularization, conjugate gradient, Newton-conjugate gradient, and L-BFGS are displayed in Tables 1 through 5. The latter three algorithms were run using the SciPy implementations.

Examining Table 1, we can see that there are several times that cubic regularization and adaptive cubic regularization did not converge after 10,000 iterations. Given that these are very non-convex problems for which the Hessian may not be globally Lipschitz continuous, this is not too surprising. However, on the BROWNAL problem, adaptive cubic regularization actually got close to the optimum; it reached a function value of 0.00011 when the optimal value was zero. Aside from these cases, cubic regularization tends to be competitive with the other algorithms in terms of the number of outer iterations (defined by each time the algorithms take a step). Similarly, the number of function and gradient evaluations, shown in Tables 2 and 3, also seem to be competitive on most of the problems. On the other hand, adaptive cubic regularization using Broyden updates seems to take many more iterations than cubic regularization, making it not competitive.

Table 1: Number of iterations until convergence for each problem by algorithm for $N = 100$. Asterisks denote a failure to converge after 10,000 iterations.

	Cubic Reg	Adaptive	CG	Newton-CG	L-BFGS
BROWNAL	7	10000*	2	2	8
BRYBND	49	878	50	17	60
DQDRTIC	1	6	5	7	11
FLETCHBV	10000*	10000*	2676	10000*	117
FLETCHCR	382	2709	1*	190	496
GENHUMPS	10000*	10000*	1669	9591	973
GENROSE	190	1065	474	91	261
MANCINO	71	85	10	7	10
MOREBV	1	2	311	5	92
SENSORS	22	155	27	20	19

Table 2: Number of function evaluations for each problem by algorithm for $N = 100$. Asterisks denote a failure to converge after 10,000 iterations.

	Cubic Reg	Adaptive	CG	Newton-CG	L-BFGS
BROWNAL	15	10017*	30	4	13
BRYBND	99	922	88	20	65
DQDRTIC	5	8	13	9	17
FLETCHBV	20001*	10002*	4366	15030*	165
FLETCHCR	765	3069	17*	221	597
GENHUMPS	20005*	10010*	2775	10251	1321
GENROSE	381	1217	732	129	328
MANCINO	143	103	22	9	15
MOREBV	3	4	467	7	97
SENSORS	48	162	54	28	22

One of the main differences between cubic regularization and the other algorithms is the number of necessary Hessian evaluations, which are displayed in Table 4. As we can see, cubic regularization performs many more Hessian evaluations than Newton-CG, which definitely slows it down. On the other hand, adaptive cubic regularization only evaluates the Hessian during initialization (in part for input checking) and when solving the cubic subproblem fails for the approximate Hessian.

Since adaptive cubic regularization evaluates the Hessian so many fewer times, we might expect that its running time would be smaller than that of cubic regularization on these problems of size $N = 100$. However, as we can see from Table 5, that is clearly not the case, and this is due to the fact that it takes so many more iterations to converge. This is likely why Cartis et al. (2011) did not present results from using an approximate Hessian. It would be interesting though to see if on even larger problems adaptive cubic regularization would take less time to converge.

Note that since the SciPy algorithms are highly optimized and are probably not written

Table 3: Number of gradient evaluations for each problem by algorithm for $N = 100$. Asterisks denote a failure to converge after 10,000 iterations.

	Cubic Reg	Adaptive	CG	Newton-CG	L-BFGS
BROWNAL	9	10002*	17	4	12
BRYBND	51	880	87	35	64
DQDRTIC	4	8	12	14	16
FLETCHBV	10002*	10002*	4353	25024*	164
FLETCHCR	384	2711	4*	409	596
GENHUMPS	10002*	10002*	2774	19837	1320
GENROSE	192	1067	731	218	327
MANCINO	73	87	21	14	14
MOREBV	3	4	466	10	96
SENSORS	24	157	53	46	21

Table 4: Number of Hessian evaluations for each problem by algorithm for $N = 100$. Asterisks denote a failure to converge after 10,000 iterations.

	Cubic Reg	Adaptive	CG	Newton-CG	L-BFGS
BROWNAL	11	2*	0	2	0
BRYBND	53	4	0	17	0
DQDRTIC	6	2	0	7	0
FLETCHBV	10004*	2	0	10000*	0
FLETCHCR	386	3	0	190	0
GENHUMPS	10004*	2	0	9591	0
GENROSE	194	3	0	91	0
MANCINO	75	3	0	7	0
MOREBV	5	2	0	5	0
SENSORS	26	3	0	20	0

in pure Python, it is not fair to compare the running times of my implementations of the cubic regularization algorithms to those of the algorithms in SciPy. Ignoring the fact that they are so much slower, it appears as though cubic regularization is competitive with the other algorithms, while adaptive cubic regularization using Broyden updates is not. It is interesting to note that the results presented here are very different from those of Cartis et al. (2011) in terms of the number of iterations until convergence for these problems, with some being significantly lower and some being significantly higher.

4 Conclusion

With the cubic regularization algorithm of Nesterov and Polyak (2006) and the adaptive cubic regularization algorithm of Cartis et al. (2011) being seemingly underutilized given their nice properties, it seemed appropriate to provide an open-source implementation of them for others to discover and use. In this report I demonstrated that my implementations

Table 5: Time until convergence for each problem by algorithm for $N = 100$. Asterisks denote a failure to converge after 10,000 iterations.

	Cubic Reg	Adaptive	CG	Newton-CG	L-BFGS
BROWNAL	0.509	79.692*	0.006	0.124	0.005
BRYBND	0.535	8.815	0.016	0.011	0.013
DQDRTIC	0.006	0.084	0.003	0.006	0.003
FLETCHBV	39.795*	143.715*	0.270	3.308*	0.010
FLETCHCR	6.305	20.934	0.004	0.084*	0.025
GENHUMPS	28.232*	123.205*	0.287	5.259	0.133
GENROSE	1.619	13.759	0.043	0.048	0.020
MANCINO	6.439	28.034	0.112	0.570	0.085
MOREBV	0.006	0.019	0.026	0.026	0.011
SENSORS	0.511	2.685	0.351	0.448	0.114

do work and I compared the number of iterations, number of function, gradient, and Hessian evaluations, and their running times on ten different functions to those of other common algorithms. The results suggested that cubic regularization performs competitively with the others, while adaptive cubic regularization using a Powell-symmetric-Broyden update for the Hessian is much slower.

Future work related to these algorithms could involve making the implementations more efficient. This could mean rewriting them in a faster language such as C or Fortran and/or using a different algorithm to solve the cubic subproblem. In addition, it would also be interesting to build on this code to implement the algorithm of Anandkumar and Ge (2016).

References

Anandkumar, A. and Ge, R. (2016). Efficient approaches for escaping higher order saddle points in non-convex optimization. *arXiv preprint arXiv:1602.05908.*

Cartis, C., Gould, N. I., and Toint, P. L. (2011). Adaptive cubic regularisation methods for unconstrained optimization. part i: motivation, convergence and numerical results. *Mathematical Programming*, 127(2):245–295.

Conn, A. R., Gould, N. I., and Toint, P. L. (2000). *Trust region methods*, volume 1. Siam.

Gould, N. I., Orban, D., and Toint, P. L. (2003). CUTEr and SifDec: A constrained and unconstrained testing environment, revisited. *ACM Transactions on Mathematical Software (TOMS)*, 29(4):373–394.

Nesterov, Y. and Polyak, B. T. (2006). Cubic regularization of newton method and its global performance. *Mathematical Programming*, 108(1):177–205.

A Additional tables

Table 6: Functions from the CUTEr test set of Gould et al. (2003) on which I tested cubic and adaptive cubic regularization for $N = 100$

Name	Function
Brownal	$\sum_{i=1}^{N-1} \left(x_i + \sum_{j=1}^{N} x_j - (N+1)\right)^2 + (\prod_{j=1}^{N} x_j - 1)^2$
Brybnd	$\sum_{i=1}^{N} \left(x_i(2+5x_i^2) + 1 - \sum_{j \in \mathcal{J}_i} x_j(1+x_j)\right)^2$
	where $\mathcal{J}_i \equiv \{j \in 1, 2, \ldots, N : j \neq i \text{ and } \max(1, i-5) \leq j \leq \min(N, i+\mu)\}$
Dqdrtic	$\sum_{i=1}^{N-2} \left(100x_{i+1}^2 + 100x_{i+2}^2 + x_i^2\right)$
Fletchbv	$0.5x_1^2 + \sum_{i=1}^{N-1} 0.5(x_i - x_{i+1})^2 + 0.5x_N^2 + \sum_{i=1}^{N} \left(-1 - 2\left(N+1\right)^2\right) x_i - \sum_{i=1}^{N}(\cos(x_i)(N+1)^2$
Fletchcr	$\sum_{i=1}^{N-1} 100(x_{i+1} - x_i + 1 - x_i^2)^2$
Genhumps	$\sum_{i=1}^{N-1} \left(\sin^2(2x_i)\sin^2(2x_{i+1}) + 0.05(x_i^2 + x_{i+1}^2)\right)$
Genrose	$1 + \sum_{i=2}^{N} 100\left(x_i - x_{i-1}\right)^2 + \sum_{i=2}^{N}(x_i - 1)^2$
Mancino	$\sum_{i=1}^{N}[1400x_i + (i-50)^3 + \sum_{j=1}^{N} v_{ij}((\sin(\log(v_{ij})))^5 + \cos(\log(v_{ij})))^5)]^2$
	where $v_{ij} = \sqrt{x_i^2 + \frac{i}{j}}$
Morebv	$\sum_{i=1}^{N} \left(2x_i - x_{i-1} - x_{i+1} + \frac{1}{2}\left(\frac{1}{N+1}\right)^2 (x_i + \frac{i}{N+1} + 1)^3\right)^2$
	where $x_0 \equiv 0$, $x_{N+1} \equiv 0$
Sensors	$\sum_{i,j=1}^{N} - \left[\sin\left(x_i\right)\sin\left(x_j\right)\sin\left(x_i - x_j\right)\right]^2$

Table 7: Starting points for the functions from the CUTEr test set of Gould et al. (2003) on which I tested cubic and adaptive cubic regularization for $N = 100$. Unless otherwise specified, the values are for $i = 1, \ldots, N$.

Name	Staring point
Brownal	$x_i = \frac{1}{2}$
Brybnd	$x_i = -1$
Dqdrtic	$x_i = 3$
Fletchbv	$x_i = \frac{i}{N+1}$
Fletchcr	$x_i = 0$
Genhumps	$x_1 = -506, x_i = 506.2, i \neq 1$
Genrose	$x_i = \frac{1}{N+1}$
Mancino	$x_i = -8.710996 \times 10^{-4}((i-50)^3 + \sum_{j=1}^{N} \sqrt{\frac{i}{j}}((\sin(\log(\sqrt{\frac{i}{j}})))^5 + (\cos(\log(\sqrt{\frac{i}{j}})))^5))$
Morebv	$x_i = \frac{i}{N+1}\left(\frac{i}{N+1} - 1\right)$
Sensors	$x_i = \frac{1}{N}$