IOE 511 – Continuous Optimization Methods

An Investigation of the Performance of Unconstrained Nonlinear Optimization Algorithms

Team: Northwood

Ram Padmanabhan
rampad@umich.edu

Shreyas Bhat
shreyasb@umich.edu

---

1 Summary of Algorithms

The goal of unconstrained optimization is to solve the following problem:

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

where $f : \mathbb{R}^n \to \mathbb{R}$. This is usually accomplished by iterative methods of the following form:

$$x_{k+1} = x_k + \alpha_k d_k, \tag{2}$$

where α_k is a (possibly varying) step size, and d_k is a search direction. A common requirement on the search direction is that it must be a descent direction, i.e. $\nabla f(x_k)^T d_k < 0$, where $\nabla f(x_k)$ denotes the gradient of f at x_k. The algorithms investigated in this project are summarized below.

1.1 Gradient Descent

Gradient descent is possibly the simplest unconstrained optimization algorithm. The search direction d_k is simply set to $-\nabla f(x_k)$, so that $\nabla f(x_k)^T d_k = -\left\|\nabla f(x_k)\right\|^2 < 0$. At each iteration, a step is taken along the negative gradient of f at the current iterate, and the length of the step is determined by α_k. Gradient descent is usually implemented using a line search. On general non-convex problems, gradient descent converges to a stationary point, but this need not even be a local minimum, and can be a saddle point.

1.2 Newton's Method

Newton's method exploits second-order information of f, such that the search direction d_k satisfies $\nabla^2 f(x_k) d_k = -\nabla f(x_k)$, where $\nabla^2 f(x_k)$ denotes the Hessian of f at x_k. If $\nabla^2 f(x_k)$ is positive definite, d_k is a descent direction. This 'Newton' direction imposes a degree of scale in a local neighborhood of the optimal point x^*. Newton's method converges quadratically to the optimal point when the initial iterate is in a certain neighborhood of this point. However, outside this neighborhood, it is possible that d_k is an ascent direction. Using a line search implementation, Newton's method can be 'globalized', provided that the Hessian is modified to ensure that d_k is a descent direction at each iteration.

1.3 Broyden-Fletcher-Goldfarb-Shanno (BFGS) and Davidon-Fletcher-Powell (DFP) Algorithms

Quasi-Newton methods such as BFGS and DFP present a convenient middle-ground between the computationally cheap gradient descent, and the fast convergence achieved in Newton's method. In both

algorithms, the search direction $d_k = H_k \nabla f(x_k)$, where H_k is an approximation of the inverse Hessian $\left[\nabla^2 f(x_k)\right]^{-1}$. At each iteration, this approximation is updated using first-order information only. Both BFGS and DFP are 'rank-2 updates', in that the the approximations are updated by rank 2 matrices, i.e. $H_{k+1} = H_k +$ a rank-2 matrix. The two algorithms are 'duals' of each other – the inverse Hessian update in BFGS has the same structure as the Hessian update in DFP, and vice versa. Both algorithms can be used in either a line search or a trust region framework. In the latter scenario, the update equations are used to modify the model in the trust region subproblem from one iteration to the next.

1.4 Trust Region – Conjugate Gradient Methods

Trust region methods differ from line search methods in that we first set a radius within which we "trust" our model of the objective function to be accurate enough. It is after this step that we compute a direction of descent on the model. To determine this direction, we solve a subproblem which tries to minimize the quadratic approximation around the current point,

$$\min_{d \,:\, ||d||_2 \leq \Delta_k} f(x_k) + \nabla f(x_k)^T d + d^T B_k d$$

Here, B_k is the true Hessian when we are using Newton's method and is the symmetric rank-1 approximation to the Hessian when we are using the SR-1 method. We solve this subproblem via the CG Steihaug algorithm. It is not necessary for B_k to be positive definite. At every iteration, we compute the ratio of the reduction in objective function value and the model value (ρ) and update our iterate based on this ratio. If the ratio is too small, we cannot trust our model within the current Δ_k radius, so we shrink the radius to half its current value and skip the iterate update. If the ratio is big enough, we update our iterate to $x_k + d_k$. If the ratio is bigger than the parameter `c2_tr`, we double the trust region radius for the next iteration.

2 Default Parameters

Table 1 provides the different default parameters for our code, which are within the structure `options`.

There are other parameters that cannot be directly modified from the structure `options`, but have to be

Table 1: Default Parameters

Description	Variable	Value
Termination Tolerance ϵ	`term_tol`	1e–6
Maximum Iterations	`max_iterations`	1e+3
Line Search Constant c_1	`c1_ls`	1e–4
Line Search Constant c_2	`c2_ls`	0.9
Trust Region Constant c_1	`c1_tr`	1e–4
Trust Region Constant c_2	`c2_tr`	0.9
Conjugate Gradient Termination Tolerance	`term_tol_CG`	1e–6
Conjugate Gradient Maximum Iterations	`max_iterations_CG`	1e+3
Cholesky Subroutine Parameter β	`beta`	1e–6
Hessian Update Threshold Parameter ϵ	`eps`	1e–6
Initial Trust Region Radius Δ_0	`delta`	10.0

modified within the functions using them. These parameters include the initial step length $\bar{\alpha} = 1$ in a backtracking line search, the backtracking parameter $\tau = 0.5$, the lower and higher limits $\alpha_{low} = 0$ and $\alpha_{high} = 1000$ in the weak Wolfe line search, and the constant c used to determine $\alpha = c\alpha_{low} + (1 - c)\alpha_{high}$ in the weak Wolfe line search.

3 Summary of Results

Tables 2a and 2b summarize the results of our experiments, in terms of number of iterations, function evaluations, gradient evaluations and CPU seconds to solve each problem, using each algorithm. The fastest and slowest algorithms for a given problem are highlighted in green and red respectively, unless the difference between these times is too small to be truly indicative of speed. Recall here that an algorithm either runs for `max_iterations` = 1000 iterations, or terminates when $\|\nabla f(x_k)\|_\infty \leq \epsilon \max\{1, \|\nabla f(x_0)\|_\infty\}$, where ϵ = `term_tol` = 1e–6. An algorithm running for 1000 iterations indicates slow convergence, or the algorithm being unable to solve that particular problem.

Further, Figures 1, 2 and 3 demonstrate the performance of all algorithms on three broadly representative problems from the given set: (i) the 1000–dimensional quadratic problem with $\kappa = 1000$, i.e. a highly ill-conditioned problem, (ii) the classic Rosenbrock problem, a well-known non-convex function frequently used to benchmark unconstrained optimization algorithms, and (iii) the 5–dimensional Genhumps function, another non-convex function with numerous 'humps', making optimization difficult. The metric chosen for comparison is the profile of gradient norm with iterations, as the optimal solution f^* may not be known for all problems. On a general non-convex problem, the best possible outcome is convergence to a stationary point, i.e. $\lim_{k\to\infty} \|\nabla f(x_k)\|_\infty = 0$. While this isn't possibly the most accurate metric, comparisons in terms of CPU time are readily available in Tables 2a and 2b.

3.1 Discussion

We now discuss a few trends in Tables 2a and 2b. It must be noted that in general, the algorithms do not struggle to converge to a stationary point. Many algorithms on many problems terminate within 200 iterations, and frequently do so within 0.20 seconds. In particular, Newton's method, the two trust region variants, and BFGS are highly reliable on any problem in the given set.

3.1.1 Difficulties

On the four quadratic problems, the condition number significantly affects the performance of algorithms. In particular, gradient descent and DFP struggle on ill-conditioned problems, and do not converge even after 1000 iterations. Non-convexity also significantly impacts convergence. On the 2–dimensional Rosenbrock problem, gradient descent and DFP with a Wolfe line search do not converge, and on the Genhumps 5–dimensional problem, DFP does not converge. This trend is however not obvious in the Rosenbrock 100–dimensional problem, possibly because in this large scale, the starting point is very close to the optimal solution, and the same 'valleys' and 'hills' on the 2–dimensional problem are not present here.

These difficulties can also be seen in Figures 1, 2 and 3. Note how gradient descent and DFP (with both a backtracking and a Wolfe line search) are not even close to convergence on the 1000–dimensional quadratic with $\kappa = 1000$ (see Fig. 1), whereas all other algorithms have terminated before 400 iterations are complete.

Table 2a: Summary of Results (1)

	Metric	GD	GDW	Newton	NewtonW	TRNewtonCG	TRSR1CG	BFGS	BFGSW	DFP	DFPW
P1_quad_10_10	Iterations	119	119	1	1	3	10	26	26	39	39
	CPU Time	0.15 s	0.16 s	0.06 s	0.06 s	0.06 s	0.06 s	0.08 s	0.07 s	0.07 s	0.08 s
	Fn. Evaluations	238	238	1	2	6	20	52	52	78	78
	Grad. Evaluations	119	238	2	2	3	10	26	52	39	78
P2_quad_10_1000	Iterations	1000	1000	1	1	9	15	55	55	1000	1000
	CPU Time	0.75 s	1.65 s	0.06 s	0.06 s	0.07 s	0.06 s	0.12 s	0.12 s	0.72 s	0.95 s
	Fn. Evaluations	2000	4400	1	2	18	27	110	110	2000	2000
	Grad. Evaluations	1000	2200	2	2	9	17	55	9	1000	2000
P3_quad_1000_10	Iterations	114	114	1	1	5	22	31	31	44	44
	CPU Time	4.44 s	4.52 s	4.40 s	4.40 s s	4.73 s	4.97 s	5.45 s	5.47 s	4.59 s	4.68 s
	Fn. Evaluations	228	228	1	2	10	44	62	62	88	88
	Grad. Evaluations	114	228	2	2	5	22	31	62	44	88
P4_quad_1000_1000	Iterations	1000	1000	1	1	10	370	357	357	1000	1000
	CPU Time	5.45 s	6.78 s	4.30 s	4.30 s	5.30 s	29.79 s	16.90 s	17.58 s	10.94 s	11.33 s
	Fn. Evaluations	2000	4364	1	2	20	578	714	714	2000	2000
	Grad. Evaluations	1000	2197	2	2	10	208	357	714	1000	2000
P5_quartic_1	Iterations	2	2	2	2	2	3	3	3	3	3
	CPU Time	0.06 s	0.06 s	0.06 s	0.06 s	0.06 s	0.09 s	0.06 s	0.06 s	0.06 s	0.06 s
	Fn. Evaluations	4	4	2	4	4	6	6	6	6	6
	Grad. Evaluations	2	4	4	4	2	3	3	6	3	6
P6_quartic_2	Iterations	6	6	12	12	12	27	24	24	65	65
	CPU Time	0.06 s	0.06 s	0.06 s	0.06 s	0.06 s	0.06 s	0.06 s	0.06 s	0.06 s	0.06 s
	Fn. Evaluations	93	174	12	24	24	48	110	172	191	252
	Grad. Evaluations	6	12	24	24	12	21	24	48	65	130

Table 2b: Summary of Results (2)

	Metric	GD	GDW	Newton	NewtonW	TRNewtonCG	TRSR1CG	BFGS	BFGSW	DFP	DFPW
Rosenbrock_2	Iterations	1000	1000	20	20	35	146	33	33	47	1000
	CPU Time	0.20 s	0.22 s	0.12 s	0.12 s	0.13 s	0.12 s	0.13 s	0.13 s	0.13 s	0.18 s
	Fn. Evaluations	10833	19764	20	40	54	221	85	104	113	2262
	Grad. Evaluations	1000	2001	47	54	19	75	33	66	47	2017
Rosenbrock_100	Iterations	42	42	4	4	4	67	112	112	108	108
	CPU Time	0.12 s	0.18 s	0.12 s	0.12 s	0.12 s	0.13 s	0.16 s	0.18 s	0.12 s	0.13 s
	Fn. Evaluations	503	922	4	8	8	94	1123	2022	721	1226
	Grad. Evaluations	42	84	8	8	4	27	112	224	108	216
Datafit_2	Iterations	532	532	6	6	15	19	14	14	21	21
	CPU Time	0.18 s	0.18 s	0.13 s	0.13 s	0.13 s	0.12 s	0.12 s	0.12 s	0.11 s	0.16 s
	Fn. Evaluations	3449	5834	6	12	24	29	36	44	50	58
	Grad. Evaluations	532	1064	12	12	9	10	14	28	21	42
Exponential_10	Iterations	21	17	13	13	12	20	10	9	10	9
	CPU Time	0.07 s	0.05 s	0.06 s	0.06 s	0.06 s	0.05 s	0.07 s	0.05 s	0.06 s	0.05 s
	Fn. Evaluations	44	52	13	26	24	35	23	36	23	36
	Grad. Evaluations	21	35	43	60	12	15	10	19	10	19
Exponential_1000	Iterations	21	17	13	13	12	15	10	9	10	9
	CPU Time	0.05 s	0.06 s	0.05 s	0.05 s	0.05 s	0.05 s	0.05 s	0.06 s	0.06 s	0.06 s
	Fn. Evaluations	44	52	13	26	24	24	23	36	23	36
	Grad. Evaluations	21	35	43	60	12	9	10	19	10	19
Genhumps_5	Iterations	147	143	96	39	131	104	95	52	1000	1000
	CPU Time	0.08 s	0.06 s	0.06 s	0.05 s	0.06 s	0.06 s	0.05 s	0.05 s	0.17 s	0.16 s
	Fn. Evaluations	362	634	96	91	201	163	234	156	2053	2270
	Grad. Evaluations	147	289	233	236	70	59	95	107	1000	2022

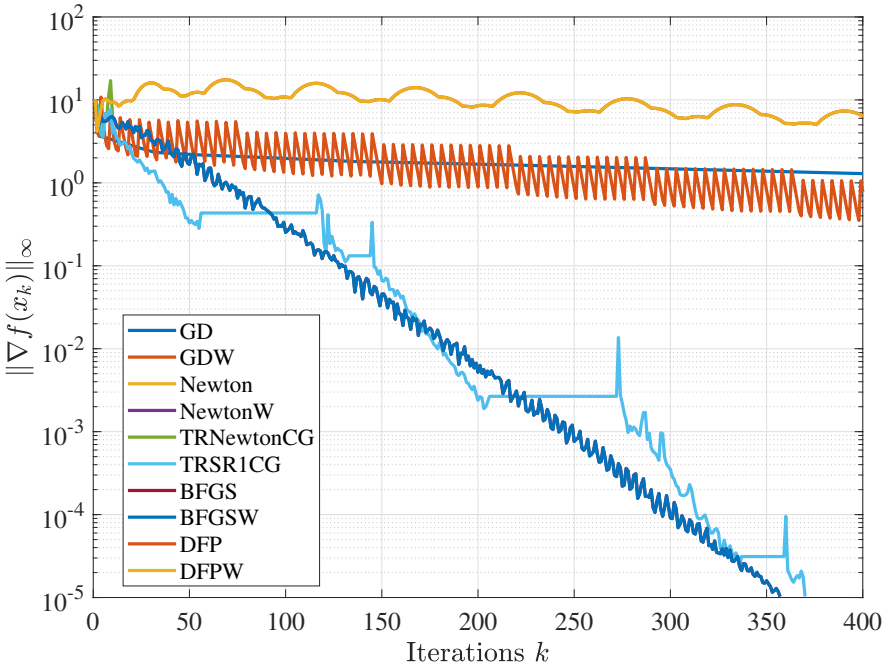

Figure 1: Performance of all methods on the 1000–dimensional quadratic problem with $\kappa = 1000$.

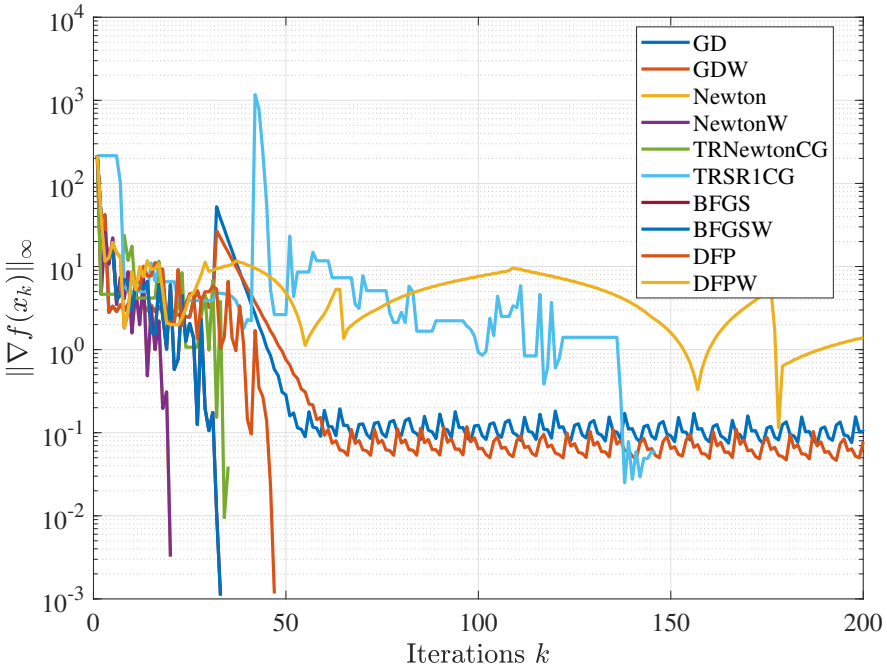

Figure 2: Performance of all methods on the Rosenbrock 2–dimensional problem

Similarly, in Fig. 2, gradient descent and DFP with a Wolfe line search are the only algorithms that are yet to terminate after 200 iterations. The more 'reliable' algorithms, including BFGS and Newton's method, terminate in less than 50 iterations. In Fig. 3, DFP is yet to terminate after 200 iterations, and

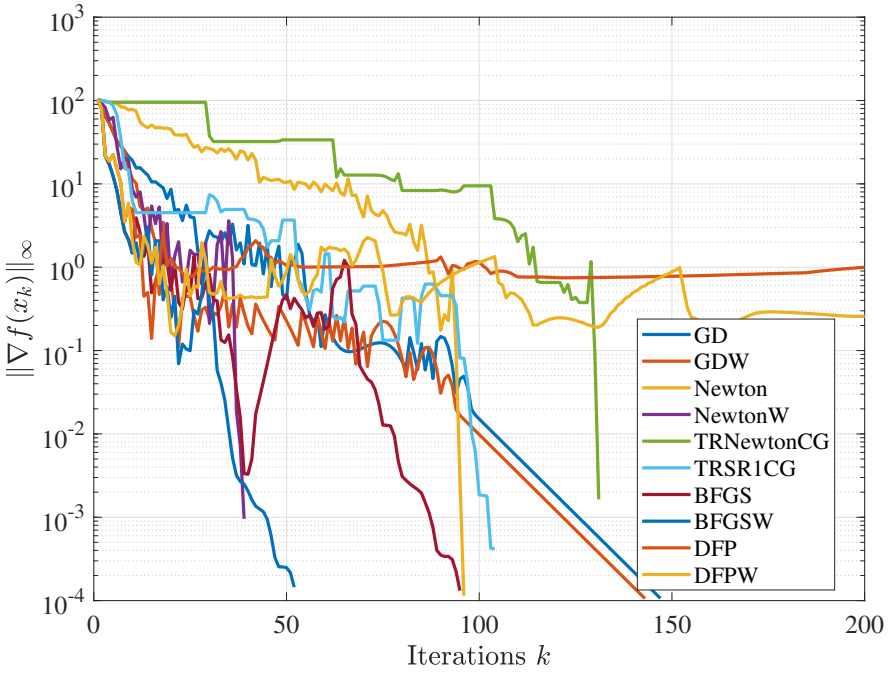

Figure 3: Performance of all methods on the Genhumps 5–dimensional problem

gradient descent takes the most iterations following that.

The reason for the difficulties in gradient descent are quite clear. The condition number directly determines the rate of convergence of gradient descent for strongly convex problems, and non-convexity means that gradient descent may struggle to find a 'good' descent direction. However, the reasons for difficulties in DFP are not so clear. One possible explanation is that unlike BFGS, which minimizes the variation in the inverse Hessian approximation from one iteration to the next, DFP minimizes the variation in the actual Hessian approximation. There is no control over what the resulting inverse Hessian approximation may be, and this could lead to a less reliable algorithm. In particular, the inverse Hessian approximation may differ drastically from one iteration to the next, especially if the problem is ill-conditioned, thus directly affecting descent directions.

3.1.2 Is there a *"Best"* Algorithm?

While no algorithm can claim to be the "best" unconstrained optimization algorithm on any problem, it is evident that certain algorithms strongly outperform other algorithms, especially on this given set. As mentioned earlier, Newton's method, the two trust region variants, and BFGS are significantly more reliable than gradient descent and DFP. Among these, **Newton's method with a Wolfe line search** (with the Hessian modification to ensure positive definiteness) frequently converges faster than the others. Despite the cost of solving a linear system in each iteration, this method is among the fastest algorithms for each problem. On all quadratics, irrespective of dimension or condition number, Newton's method converges in a single iteration. Even on high-dimensional problems such as the 100–dimensional Rosenbrock or the 1000–dimensional exponential, the Hessian modification ensures fast, global convergence, in terms of both iterations and CPU time. On all the three representative problems mentioned earlier — (i) the

1000–dimensional quadratic problem with $\kappa = 1000$, (ii) the 2–dimensional Rosenbrock problem, and (iii) the 5–dimensional Genhumps problem, Newton's method with a Wolfe line search converges fastest in terms of both iterations and CPU time. This is hence the "algorithm of choice".

4 The Hessian Modification in Newton's Method

This section is devoted to investigating the following question: **"How much modifying is too much in Newton's method?"**

Recall that the search direction d_k in Newton's method satisfies $\nabla^2 f(x_k) d_k = -\nabla f(x_k)$. If $\nabla^2 f(x_k)$ is positive definite, then the above linear system has a unique solution d_k, and this d_k is a descent direction. However, when $\nabla^2 f(x_k)$ is not positive definite, there may be multiple solutions to the linear system, and/or d_k may not be a descent direction. To mitigate this, a multiple of the identity is added to the Hessian at each iteration, so that the linear system now becomes $\left(\nabla^2 f(x_k) + \eta_k I\right) d_k = -\nabla f(x_k)$. η_k is chosen such that the matrix $\left(\nabla^2 f(x_k) + \eta_k I\right)$ is positive definite, and hence d_k is unique and is a descent direction. This is accomplished by attempting a Cholesky factorization of $\left(\nabla^2 f(x_k) + \eta_k I\right)$ multiple times, and updating η_k until the factorization is successful. (This is hereafter referred to as the Cholesky subroutine.)

The motivation for investigating the above question is as follows. If η_k is very large, it effectively ensures that d_k is unique and is a descent direction. Hence, it may be tempting to set η_k to be a large constant *a priori*, and forgo the complexity of choosing an appropriate η_k at each iteration. However, if η_k is too large, we have $\left(\nabla^2 f(x_k) + \eta_k I\right) \approx \eta_k I$, and hence the search direction is simply a scalar multiple of the negative gradient direction. While this is the steepest descent direction for our purposes, the advantages of Newton's method, such as a quadratic rate of convergence, are lost. Here, we attempt to investigate what the limits of modifying $\nabla^2 f(x_k)$ are. In particular, we focus on the 2–dimensional Rosenbrock problem and compare the rates of convergence for various increasing values of η_k. Further, we compare these rates to the rate of gradient descent, and the rate obtained by the usual modification of the Hessian, obtained from the Cholesky subroutine. We aim to find a level of values of η_k, beyond which modification of the Hessian no longer presents the same advantages as Newton's method, and begins to demonstrate disadvantages of the gradient method. The Rosenbrock problem is particularly appropriate for this experiment, as the gradient method struggles for convergence and Newton's method does not.

The methodology adopted for controlling the modification was as follows. The Cholesky subroutine was run on the true Hessian at the current iteration point to ensure that the resulting matrix is positive definite. Post this subroutine, an additional λI was added to the resulting matrix where λ was the independent variable. The resulting descent direction is then given by $d_k = -\left(\nabla^2 f(x_k) + (\eta_k + \lambda) I\right)^{-1} \nabla f(x_k)$.

It was noted that the eigenvalues for the true Hessian matrix for the Rosenbrock problem for the given conditions and for all iterates visited were approximately in the interval $(5, 50)$.

Figs. 4 and 5 show the performance of the method for different values of λ. Since, the true Hessian for this problem is always positive definite for all the visited iterates, it seems that even small modifications lead to a loss in performance. Around $\lambda = 500$, we lose the quadratic rate of convergence of Newton's method to a more linear rate. Around $\lambda = 700$, we are unable to converge within 1000 iterations. And for values of λ around 1000, we tend to perform even worse than gradient descent (within the first 1000

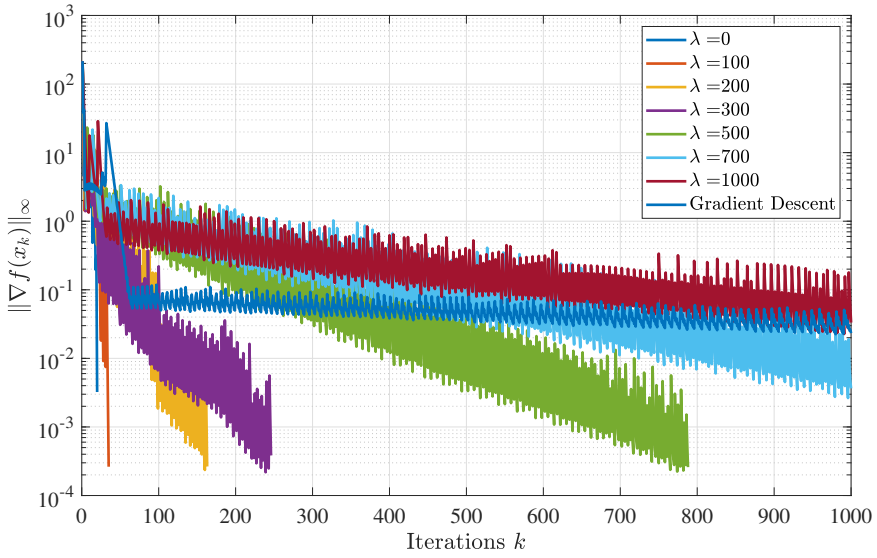

Figure 4: Performance measured by the norm of the gradient of Newton's method with varying degrees of modification and Gradient Descent on the Rosenbrock 2 problem

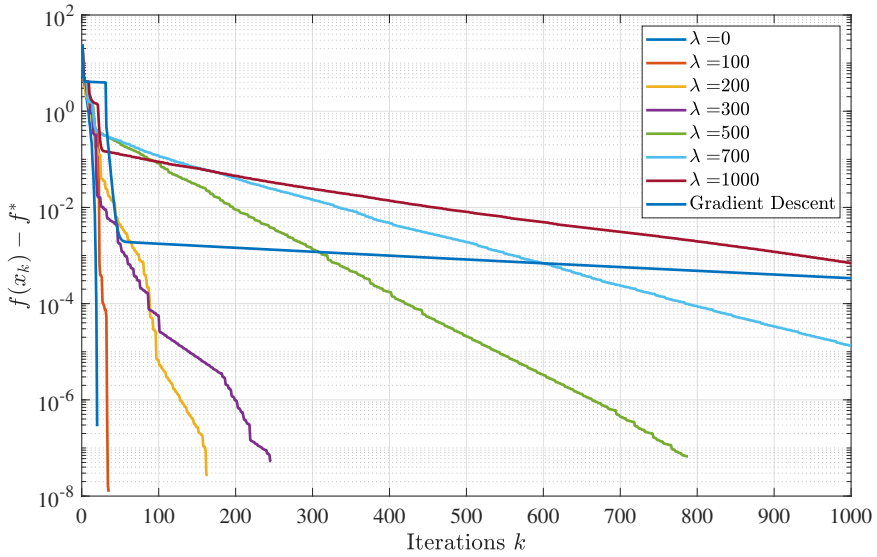

Figure 5: Performance measured by the optimality of Newton's method with varying degrees of modification and Gradient Descent on the Rosenbrock 2 problem

iterations), with a very flat-looking optimality gap plot.

However, it is worth noting that as λ is increased, the CPU time to solve the problem (or run 1000 iterations) decreases, as can be seen in Fig. 6. This can be attributed to the Hessian becoming more and more closer to a multiple of the identity matrix, thus making it easier to invert.

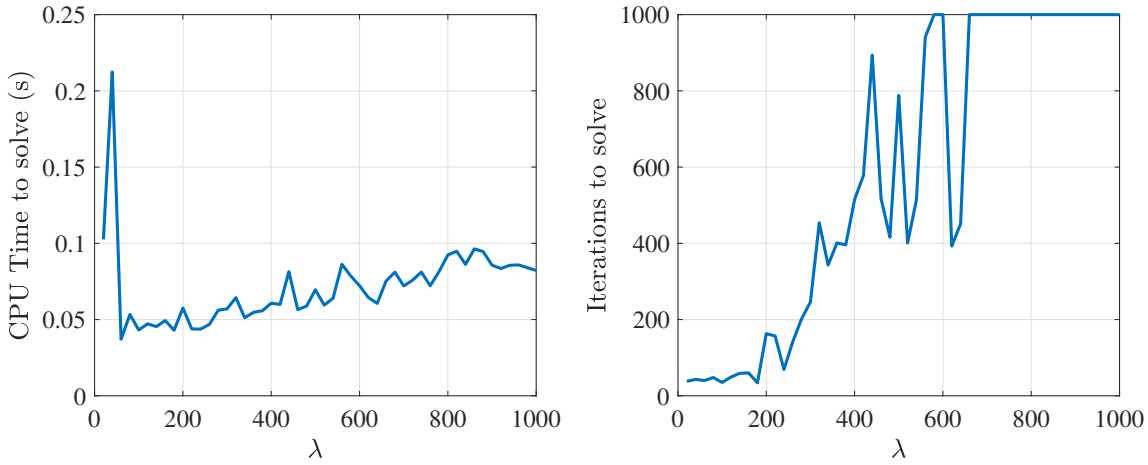

Figure 6: CPU time and number of iterations to solve the Rosenbrock 2 problem with Newton's method with different degrees of modification

---

5 Concluding Remarks

A few concluding remarks are in order. As mentioned earlier, Newton's method with a Wolfe line search generally outperforms other algorithms, and is more reliable on this particular set of problems. It must be noted that this is not a guarantee that this particular method will work well on any problem. Newton's method is significantly affected by scaling, and on problems with much higher dimensions, Newton's method is quite computationally expensive, and hence slow in terms of CPU time.

Gradient descent is possibly the easiest algorithm to code, even if used with a line search. Newton's method is not much harder, but requires second derivatives. The trust region methods and quasi-Newton methods are a little harder to code, as they require updating multiple parameters in each iteration. However, these methods are also quite reliable, and possibly scale better than Newton's method with problem dimension.