Weihui Gu(whgu@umich.edu) Tunan Wang(wtunan@umich.edu) Jiayao yang(jiayao@umich.edu)
Team OBT (Optimization Big Three)

IOE 511/MATH 562: Project
An investigation of the performance of unconstrained optimization algorithms and different line search methods

May 19, 2022

1 Introduction

In this final project, we implemented 10 unconstrained optimization algorithms to solve 12 different problems. The methods can generally be classified into two main categories: (a) line search type and (b) trust region type. For the trust region type algorithm, we used a conjugate gradient search method to solve the sub-problem. Based on that, we investigate how to select the For the line search type methods, we first implemented the backtracking and the Wolfe line search method. To investigate the big problem of **"Is there any other better line search methods?"**, we also implemented two different line search methods Golden section and quadratic interpolation line search. In this report, we summarized the results and performance the algorithms. In section 2, we gives a brief description to each of the method and provides the default options setup of our solvers. The performance of the algorithm is summarized in the section 3. The detail analysis and comparison of the 4 different line search methods are made in section 4.

2 Algorithm description and default option

2.1 Brief descriptions on each algorithm

A brief description of each method and its short name is provided as follow.

1. **GD**: Gradient descent with backtracking line search

   This is a line search method. The descent direction is chosen by the negative gradient direction which is the local steepest descent direction with $\|\cdot\|_2$ norm. The step size is computed by backtracking line search to ensure the decrease of the function value. The convexity does not affect the algorithm, and the algorithm decreases to a stationary point whose norm of gradient is small enough.

2. **GDw**: Gradient descent with Wolfe line search

   This is also a line search method, and it choose the negative gradient as the descent direction. To ensure the sufficient decrease, it uses the Wolfe conditions for line search which also ensures the step size satisfies the curvature condition. Comparing with the backtracking line search, the Wolfe conditions has more constraints on the step size and ensure the step size not too small in addition to the sufficient decrease.

3. **NM**: Newton method (with modified Hession) with backtracking line search

This is a line search method. Newton's method uses the quadratic model based on the Hessian of the objective function. With modified Hessian, Newton's method guarantees the descent direction. The direction are calculated by negative inversed hessian multiple gradient. And the step size is calculated by backtracking method. The convexity of Hessian is guaranteed by using Cholesky factorization to remove all the non-positive eigenvalue. This method is not suitable for solving large problem as the modification of the Hessian is expensive.

4. **NMw**: Newton method (with modified Hession) with Wolfe line search

   This is a line search method. The direction are calculated by negative inversed hessian multiple gradient. And the step size is calculated by backtracking method. With modified Hessian, Newton's method guarantees the descent direction. And with Wolfe conditions, it guarantees the global convergence. In addition, Wolfe conditions ensure the step size satisfy the curvature condition to avoid picking too small step size.

5. **BFGS**: BFGS quasi-Newton with backtracking line search

   This is a line search method, and at each iteration it approximate the objective function with a quadratic model. But local model is built by using the approximation of the inverse of the Hessian, and updated by the BFGS algorithm. The convexity will affect the algorithm because when calculating the descent directions we assume the positive definite of the Hessian. In this case, the update of the inverse of the Hessian approximation is adopted if the updating is positive definite, otherwise we will skip that update. In addition, the step size is chosen by backtracking line search to ensure the decrease of the objective and to handle the nonconvex problems.

6. **BFGSw**: quasi-Newton with Wolfe line search

   This is a line search method which use Hessian approximation to determine the descent direction. The Wolfe conditions for line search ensures the global convergence of the algorithm. In addition, using Wolfe conditions help the algorithm avoid picking step size too small.

7. **DFP**: DFP quasi-Newton with backtracking line search

   This is a line search method. The descent direction is computed by a local quadratic model with Hessian approximation. Comparing with BFGS, it updates the approximation of Hessian by DFP algorithm. And the updating of Hessian approximation is adopted when the updating is positive definite. The backtracking line search helps the algorithm to handle the nonconvex problem, and ensure the decrease of the objective function at each iteration.

8. **DFPw**: DFP quasi-Newton with Wolfe line search

   This is a line search method. Comparing with the use of backtracking, the difference is that it uses Wolfe line search to compute the step size. It also ensures the decrease of the objective function which helps handle the nonconvex problems. In addition, the Wolfe conditions ensures the step size is not to small.

9. **TRNMCG**: Trust region Newton with Conjugate Gradient (CG) subproblem solver

This is a trust region method. In each iteration, a second order model with the true Hessian is built to approximate the function. A constrained sub-problem is solved with the Conjugate Gradient to find the search direction and step size. The non-convexity of the Hessian has no effect on this method as the maximum step size is upper bounded by the radius of the trust region.

10. **TRSR1CG**: SR1 quasi-Newton with CG subproblem solver

    This is a trust region method. Everything is the same as the TRNMCG, except that a approximated Hessian obtained from symmetric rank-1 update is used in building the second order model.

2.2 Default options

The choice of parameters are important for a successful implementation of the algorithms. An good algorithm can work poorly if the parameters is not properly selected. We summarized our choice of parameters for different algorithms here which usually lead to a stable performance. We have set them as the default options of our implemented package of algorithms.

Line search methods: Many algorithms we implemented used line search methods to guarantee the global convergence. Backtracking and Wolfe conditions are two type of inexact line search method. A summary of the default options[1] and default values for these two line search methods as table 1, where c_1 is the sufficient decreasing condition's constant, c_2 is the curvature condition's constant, and α_l, α_h is the zoom parameters.

Method	Default options and default values					
	initial step size	contraction factor	c_1	c_2	α_l	α_h
Backtracking	$\alpha = 1$	$\tau = 0.5$	$c_1 = 10^{-4}$	-	-	-
Wolfe	$\alpha = 1$	$\tau = 0.5$	$c_1 = 10^{-4}$	$c_2 = 0.5$	$\alpha_l = 0$	$\alpha_h = 1000$

Table 1: Default options for inexact line search methods.

We also investigated the other line search methods in our experiments. Our package also includes two type of exact line search methods which are Golden section method and interpolation method. The detailed descriptions of these two methods are in section 5. And a summary of the default options for these two exact line search methods are in table 2, where ϵ_g is the precision of the minimum point with the search region. γ_1 and γ_2 represent the shrinking ratio of the line search region for each step, which is the analog of backtracking's τ. You can find how it works in Algorithm 3.

Method	Default options and default values				
	initial step size	precision	c_1	γ_1	γ_2
Golden section	$\alpha = 1$	$\epsilon_g = 10^{-3}$	-	-	-
Interpolation	$\alpha = 1$	-	$c_1 = 10^{-4}$	$\gamma_1 = 0.1$	$\gamma_2 = 0.5$

Table 2: Default options for other line search methods we investigated.

Different algorithms: For different algorithm, we also require some default input parameters. One type methods is based on the line search, and a summary of the default options is in table 3, where ϵ_{pd}

[1] - means this options is not required for this method.

is the a small positive number help determines whether to update the Hessian approximation for BFGS and DFP methods. β_N is the parameter used to determine whether to modify the Hessian matrix to be positive definite.

method	default options and default values	
	ϵ_{pd}	β_N
Gradient descent	-	-
Newton (modified)	-	$\beta_N = 10^{-6}$
BFGS	$\epsilon_{pd} = 10^{-6}$	-
DFP	$\epsilon_{pd} = 10^{-6}$	-

Table 3: Default options for different algorithms based on line search(exclude line search options).

Trust region: Another type of methods we implemented are trust region methods: TRNMCG and TRSR1CG. A summary of default options for these trust region method are displayed in table 4, where ρ is the rate of updating the radius of trust region, c_1 and c_2 are the threshold value for determining the performance of the TR search, H_0 is the initial Hessian matrix of TRSR1CG, and ϵ_{cg} and k_{max} are respectively the error tolerance and maximum iteration of the CG sub process.

Method	Default options and default values							
	initial radius	maximum radius	changing rate	c_1	c_2	H_0	ϵ_{cg}	k_{max}
TRNMCG	$\Delta_0 = 1$	Δ_{max}=1000	ρ=2	0.01	0.75	-	0.01	50
TRSR1CG	$\Delta_0 = 1$	Δ_{max}=1000	ρ=2	0.01	0.75	I	0.01	50

Table 4: Default options for different algorithms based on trust region.

Termination conditions: The last but also important, we also need to determine the conditions for the termination of our algorithms. The default options we use for the termination conditions in our implementation is maximum iteration $\mathbf{1000}$ and tolerance of gradient $\mathbf{10^{-6}}$.

2.3 Comments on implementation of algorithms

2.3.1 Line search and Wolfe conditions

Our package implemented the weak Wolfe conditions for line search(based on the algorithm provided from canvas). Though the Wolfe conditions can usually provide with a larger step size, the Wolfe condition is more strict than the Armijo backtracking line search. And there are cases that no step size satisfies the Wolfe conditions. So in addition to the implementation of weak Wolfe conditions, we also added a modified method of choosing initial step1. More details about why choosing this type of initial steps is provided in 3.5.

Algorithm 1 Modified method for choosing initial step

Require: Last step's initial step α

if $\alpha \geq 1$ **then**

$\alpha \leftarrow \min(2 \times \alpha, 10)$

end if

return α

2.3.2 The situations we regard as fails of the algorithm

Considering whether a problem is solved by one algorithm will depends on many factors, such how close to the optimal solution, and how long we expected an algorithm output an solution. In our experiment, we mainly interested in the number of iterations. When the total iteration number is too large, for example exceeds 1000 in our experiments, we regard it as the fails of the algorithms. Once an algorithm terminates within 1000 iterations, we say that the algorithm solves the problem.

3 Summary of Results

3.1 An overlook

We summarize all the results in table 5 by running all algorithms with the default options to solve all the problems. This table gives us a first view of our performance of our implemented algorithms on different problems. All the algorithms except the Gradient descent (GD) successfully solved all the problems with the desired accuracy in 1000 iterations. The failure of GD method is caused by its inherent linear convergence rate, which requires more iterations in solving the same problem. The GD can also solve the problem if the limitation on the iterations is removed. In subsections below, we will discuss more details about the results of different algorithms and problems. And the failure situations of GD are labeled by 'F' in the table 5.

We should also note that, the CPU time results can be not accurate here. Because Matlab have implicitly applied optimizations to the math operations, which will make the results not that informative. For example, when the NMm and NM solved a same problem using same number of iterations, NMw should be solver than NM as it need an extra gradient calculation. However, the results show that NMw was faster.

Here we make our plots using the sub-optimalty[2] as y-axis. However, in some questions, methods do not convergence to the same solution, it can be a little unfair to compare different methods' convergence rate here. Since we just concern the convergence ability of different methods, then the best we can do is using sub-optimality to measure the ability.

3.2 Analysis for specific problems

For problem 1-4, Newton's method(NM) converges fastest. This is guaranteed by the methods' properties that they will reach the local minimum in one step. Because Newton-based methods all use the true

[2]Find the lowest value achieved by different methods and use this value to be our bench mark for each plot. Then, plot each methods' function value minus this "fake" optimality.

Performance	Algorithm	Problem											
		1	2	3	4	5	6	7	8	9	10	11	12
Iterations	GD	46	1000F	1000F	1000F	2	6	1000F	42	530	18	9	32
	GDw	46	1000F	1000F	1000F	2	6	1000F	42	530	33	15	23
	NM	1	1	1	1	2	5	18	6	11	12	12	48
	NMw	1	1	1	1	2	5	17	6	11	12	12	35
	BFGS	51	54	59	300	3	19	33	112	17	19	11	97
	BFGSw	37	52	41	317	3	20	44	112	14	16	12	47
	DFP	48	161	48	541	3	11	106	44	14	14	14	37
	DFPw	44	209	42	497	3	12	145	28	21	13	12	105
	TRNMCG	8	12	11	22	3	5	31	4	7	12	13	49
	TRSR1CG	13	19	37	265	3	23	79	71	20	18	18	65
Function Evaluations	GD	91	1992	1999	1999	4	94	9834	462	2908	36	18	92
	GDw	91	2031	1999	2010	4	94	9883	462	2908	76	39	72
	NM	2	2	2	2	4	9	47	12	35	40	40	106
	NMw	2	2	2	2	4	9	49	12	35	40	40	121
	BFGS	102	105	117	599	6	97	82	1013	41	39	26	273
	BFGSw	81	113	90	1070	6	104	114	1013	46	45	37	166
	DFP	96	319	96	1062	6	38	228	97	31	30	31	80
	DFPw	96	644	91	1610	6	44	462	73	56	35	35	341
	TRNMCG	9	13	12	23	4	6	32	5	8	13	14	50
	TRSr1CG	14	20	38	266	4	24	80	72	21	19	19	66
Gradient Evaluations	GD	47	1001	1001	1001	3	7	1001	43	531	19	10	33
	GDw	47	1006	1001	1002	3	7	1002	43	531	35	17	25
	NM	2	2	2	2	3	6	19	7	12	13	13	49
	NMw	2	47	2	2	3	6	19	7	12	13	13	42
	BFGS	52	55	60	301	4	20	34	113	18	20	12	98
	BFGSw	39	55	43	372	4	22	46	113	16	19	14	53
	DFP	49	162	49	542	4	12	107	45	15	15	15	38
	DFPw	46	253	44	612	4	15	179	30	23	15	14	126
	TRNMCG	9	13	12	23	4	6	32	5	8	13	14	50
	TRSr1CG	14	20	38	266	4	24	80	72	21	19	19	66
CPU Time(s) (rounding results)	GD	7e-3	2e-2	1e+1	1e+1	1e-4	3e-4	3e-2	2e-3	2e-2	4e-4	3e-4	6e-4
	GDw	1e-2	4e-2	1e+1	1e+1	8e-5	3e-4	3e-2	3e-3	1e-2	1e-3	5e-4	5e-4
	NM	2e-3	6e-3	7e-2	7e-2	1e-3	3e-4	2e-3	3e-3	4e-3	1e-3	5e-3	6e-3
	NMw	7e-4	5e-4	8e-2	7e-2	2e-4	3e-4	1e-3	2e-3	8e-4	6e-4	4e-3	3e-3
	BFGS	5e-3	7e-3	2e+0	8e+0	2e-4	8e-4	2e-3	4e-2	7e-4	9e-4	4e-3	3e-3
	BFGSw	3e-3	5e-3	1e+0	1e+1	2e-4	1e-3	2e-3	3e-2	6e-4	8e-4	4e-3	2e-3
	DFP	5e-3	8e-3	1e+0	1e+1	2e-4	5e-4	5e-3	9e-3	5e-4	6e-4	4e-3	2e-3
	DFPw	4e-3	1e-2	1e+0	1e+1	2e-4	6e-4	6e-3	7e-3	8e-4	7e-4	4e-3	5e-3
	TRNMCG	1e-2	2e-3	3e-1	8e-1	5e-4	3e-4	1e-3	1e-3	4e-4	5e-4	2e-3	2e-3
	TRSR1CG	1e-2	3e-3	6e-1	6e+0	2e-4	9e-4	3e-3	1e-2	8e-4	8e-4	4e-3	3e-3

Table 5: Summary of results: running all problems and all algorithms

Hessian in building their 2^{nd} order model, which made them performed well. And the problem is also quadratic and makes Newton methods fit well. For example, the error between the iteration and optimal solution of problem 3 is plotted in figure 1 as a example to demonstrate their performances. And for other algorithms we can see that their performance started getting worse when the condition number becomes large.

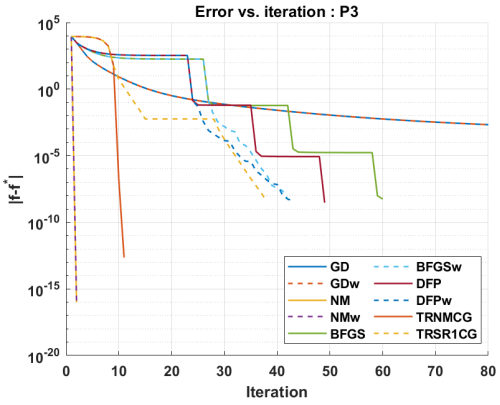

Figure 1: The decrease of function value using different algorithms on problem 3.

For problem 5 and 6, the modified Newton's method is fast. And for problem 6, the trust region Newton with CG Newton's method also works efficiently as Newton's method with modification. A comparison of performance of different algorithms is in figure 2. Note that, with the increasing of condition numbers from 5 to 6, all the quasi-newton-based methods' performance decreasing, which may consider the influence of the absence of strong-smoothness.

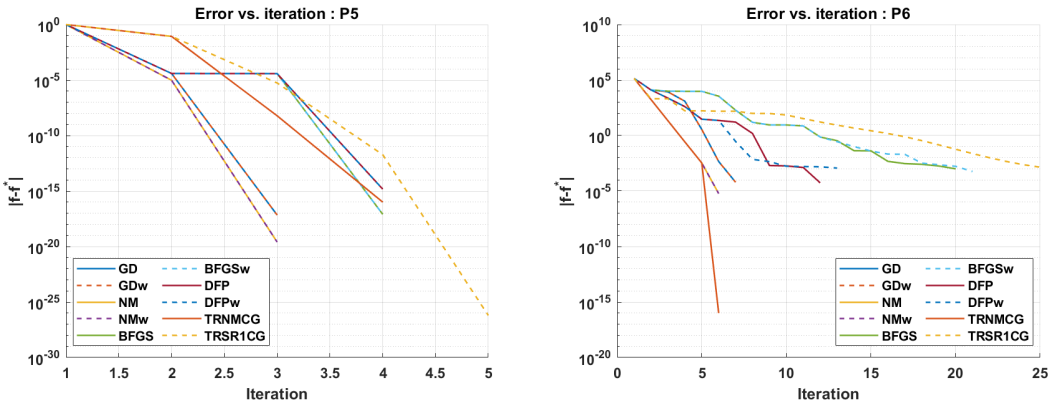

Figure 2: The decrease of function value using different algorithms on problem 5(left) and problem 6(right).

For problem 7, the NM search works best. Though NMw also gives small iterations, the number of evaluations on gradient of NM is twice smaller from the result table 5.

For problem 8-10, the TRNMCG method is better than Newton's method with line search. Though their total iterations are small, the TRNMCG give less iterations and uses less evaluations of function and gradient from the result table 5.

For problem 11 and 12, however, the GD gives the best performance, with the least iterations and small number of evaluations on function and gradient. From figure 3 and 4, we can see that all the

algorithms give similar convergence rate, but considering the time cost the GD works more efficiently on this problem. One possible explanation is that problem 12 is nonconvex, and the second order model approximation may not good enough to catch all the local information. Comparing our result in Table 5 with Table 6, we can find that our change in initial steps actually influences the most. It should be the quasi-Newton methods perform the best.

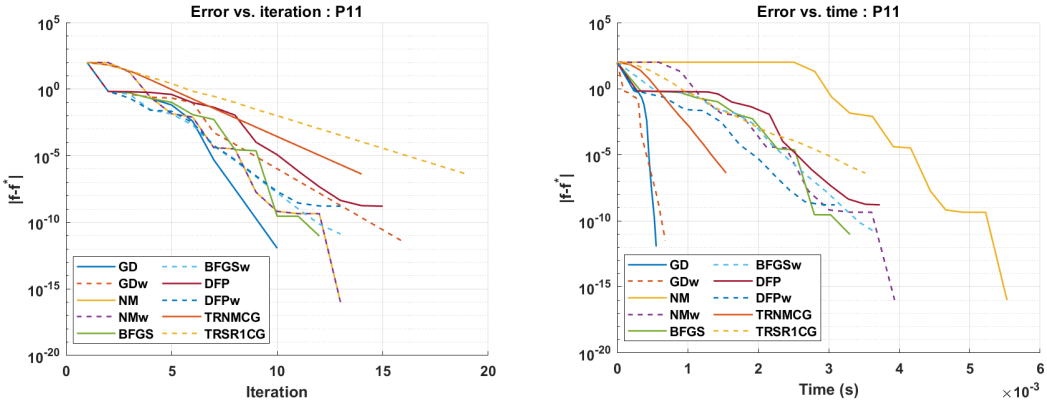

Figure 3: The iterations(left) and running time(right) of different algorithms on problem 11.

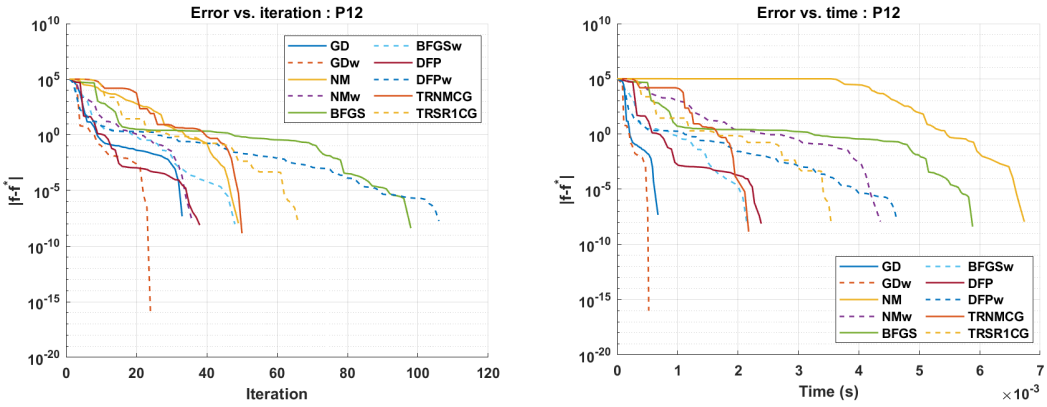

Figure 4: The iterations(left) and running time(right) of different algorithms on problem 12.

3.3 Which algorithm efficiently solves more problems?

Based on our experiments, we see that Modified Newton's method performs well on most of the problems. But, it is worth to notice that the problem we tested are either of small scale or convex with Hessian can be easily computed. Thus, we'd better say that if the scale of the problem is not large, choosing modified Newton's method will give the best performance.

To better compare the performance of different algorithm, the performance profile is plotted in Fig. 5, where we consider a problem is solved when the algorithm stopped updating. From the plot of performance profile, we can see that the Newton's method with modification can efficiently solve more problems with less iterations. The trust region method is also a good variant and gives stable performance for most problems that been tested. From right part of Fig. 5, trust region Newton method can solve more problem fast within 0.02 seconds.

However, it would be hard to decide which algorithm is the best choice for a unknown problem. **It de-**

pends. A best choice of algorithm will depends on what algorithm we are solving and what performance metric we are interested in. If we need to choose one algorithm that balances cost and converges speed, Newton's method with wolfe line search(NMw) would be a good choice for solving these 12 problems. It the fastest for most problems in our test, and it performs stable and computation cost is usually small. If the problem we want to handle is more general beyond these 12 types of problems, we would recommend trust region SR1 with CG solver(TRSR1CG) in case the Hessian information is not available. The reason that we do not recommend the modified Newton method is that it required matrix factorization, which is expensive and unaffordable for problem of large size.

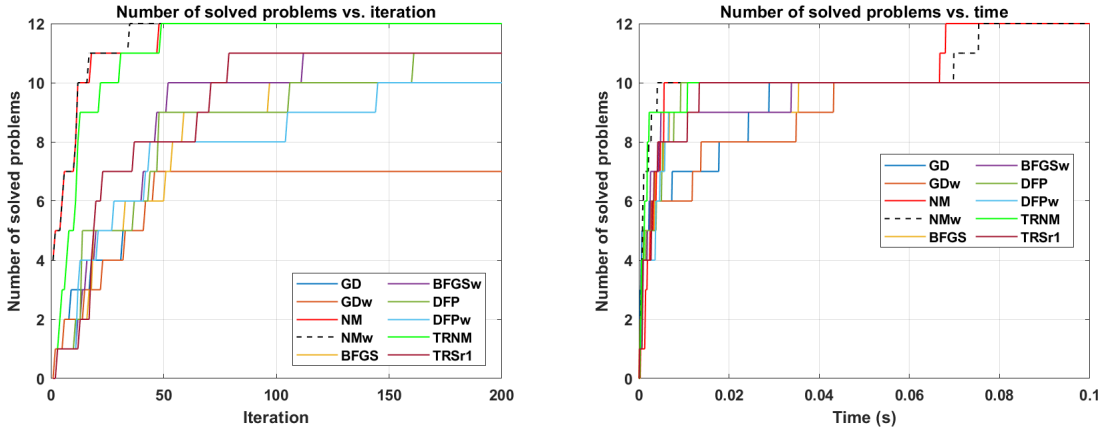

Figure 5: Performance profiles

3.4 Quasi-Newton method: BFGS vs. DFP

It is worth to mention that when implementing these two methods, the matrix and matrix multiplication should be avoided carefully, as the cost of multiplying two $n \times n$ matrix is on the order of $O(n^3)$. By properly selecting the order of multiplication, we can make sure that only the matrix and vector operations are used in the BFGS and DFP update. Two methods are pretty similar as they share the same calculation cost under each iteration and using a approximated Hessian in building the second order model. The difference between them is the BFGS gives a update on the inverse of Hessian, while the DFP updates the Hessian. Given the similarity of the two methods, we observed a quite different performances when using them to solve the same problems. While the BFGS outperformed DFP in problem $2, 4, 7$, the DFP wins in $6, 8$. For problem 12, DFP wins without Wolfe and BFGS wins with Wolfe. For these different behaviours, we should say: "**It depends!**". All the things we know is these quasi-newton methods are sensitive to the condition number, which is also the indicator of strongly-smoothness of the function, the smaller the better the performance is. It should noticed that the above analysis is based on the results with the modification made in the line search method. Without the modified we have done in line search, which will be explained below, BFGS always wins in all the problems.

3.5 Line search: Backtracking vs. Wolfe conditions

Wolfe is always better than backtracking? Yes, it is true in terms of iterations, but this is not true in terms of time or number of function or gradient evaluations. Wolfe uses an extra curvature to avoid a too small step size is produced in the the line search which is more accurate than backtracking. However,

it requires more searching sub-iterations, and more number of function and gradient evaluations, which may make Wolfe be not suitable for solving gradient-expensive problem. Moreover, although Wolfe has a theoretical guarantee to find a feasible step size if $c_1 < c_2$. In practise, we did observe that the Wolfe search failed as it was trapped in the while loop. To make the algorithm more stable, we have set a upper limit k_{max}^{wo} on the number of Wolfe search iterations, if the Wolfe fails to find the desire step size within k_{max}^{wo}, then we use the backtracking instead and a warning will be generated. It should be noticed that, when we run our solver in generating the Table 5, no warning has been generated.

Why Wolfe behaves bad here? From our table above, you may feel a little confused that why Wolfe line search cannot always lead to a better convergence performance in terms of iterations. That's because we have modified all of our line search methods to an adaptive version1. If we present our original line search methods, i.e. all the initial step size is 1, there will be a clear better performance of Wolfe than backtracking[3], at least equal to. For comparison, please find the original initial step size's result in appendix Table 6.

Why we modified line search method? The motivation of modifying the linear search method is to make DFP works for most problems. Before we modified the line search initial step size, DFP failed in most of situations, and had all the initial step size to be 1 in the last hundreds steps[4]. This indicated that the initial search step is too conservative and a dynamic selection of α_0 may help to resolve this issue. After we implementing the new initial step choosing strategy as described in 1 above, DFP works for all problems, while all the other method maintained the same performance as before.

3.6 Trust region method

How to select the initial radius Δ_0? One problem encountered in implementing the TR type method is how to select the initial radius Δ_0. It turns out that most of the tested problems are insensitive to the Δ_0, as the radius of the trust region is dynamically updated based on the performance of the last iteration. So it can quickly adjust to a reasonable size as the iteration goes on. One exception is the problem 6 (P6), which the performance of TR method has a strong dependence on the Δ_0. Specifically, the iteration used by the TR method for solving it can be five time less if we change Δ_0 from 0.01 to 1, as shown in Fig.6. One possible explanation is that a different Δ_0 will give a method to a different initial searching direction p_0, as the searching direction depends on the size of the radius especially when Δ_0 is small. At the same time, P6 is a typical rough function, so a change in the p_0 will result in a very different path to the optimal solution. The performance are almost the same if we further increases the Δ_0, which indicates that we should not set a too small Δ_0 for the Trust region type method. Based on the above test, we finally select $\Delta_0 = 1$ as the default value.

[3]Comparison happens only between different convergence

[4]DFP is oscillating near the local minimum.

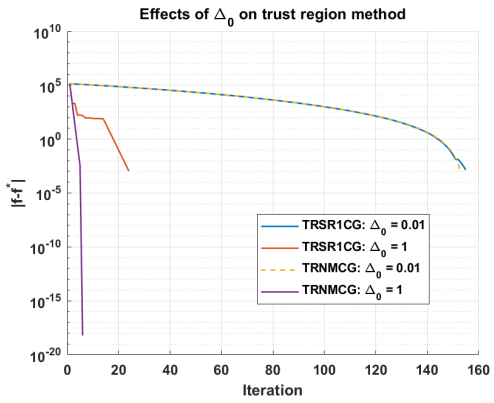

Figure 6: Effects of Δ_0 on trust region method.

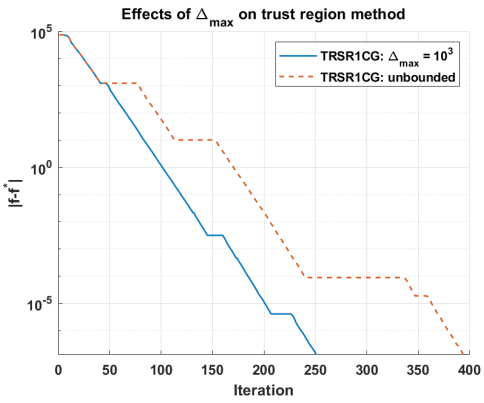

Figure 7: Training accuracy with different batch size

Should we set a upper bound for the trust region radius? As we mentioned before, the TR has a dynamic scheme in updating the size of radius. Specifically, the radius will increase $\Delta_{k+1} = 2\Delta_k$ if the model $m(p_k)$ has a good prediction, otherwise it will decrease $\Delta_{k+1} = 0.5\Delta_k$. This is a robust and self adaptive process that does not necessary need a upper bound over Δ_k. The unbounded version of the TR type solvers work well for all for the non-quadratic type problem. For quadratic type problems, the radius of the trust region can be very large as the model consistently works well. This is also not a issue for TRNMCG as it uses the true Hessian and a upper bound of the Δ_k actually will deteriorate its performance. However, a unbounded Δ_k is not a good choice for TRSR1CG. In our experiment of using TRSR1CG to solve P4, we found that the SR1 update worked well at the every beginning, which made the Δ_k very large. When the SR1 update failed to provide a good approximation, the TR method would reject the update and it stayed at the same position. As the Δ_k was a supper large value, so it toke a long time for Δ_k reduced back to a working range. That's the reason what there is flat section in Fig. 7. This situation can be fixed by setting a upper bound on Δ_k, e.g. we used $\Delta_{max} = 1000$ and the length of the flat section is significantly reduced. In the meantime, such a upper bound has a small impact on solving those non-quadratic type problems, as the radius usually is a small value.

How the CG iteration affect the TR type method? In each step, the search direction is determined by the a sub CG searching process, so the number of CG iterations play a critical role in deciding the total cost. Even though CG is a low cost and efficient method in solving the sub-problem, a too strict tolerance may dramatically increase the total cost of the TR type method. In our experiment of using TRSR1CG to solve P4, we observed that if we change the CG tolerance from $\epsilon_{cg} = 1 \times 10^{-4}$ to $\epsilon_{cg} = 1 \times 10^{-1}$, while the number of TR iteration stayed almost the same, the total number of the sub CG iterations is reduced by 10 times. Consequently the total time is reduced by 25 percent. This also demonstrated that a high accuracy solution of the sub-problem is actually not that important. Based on our numerical test, we finally select the $\epsilon_{cg} = 1 \times 10^{-2}$ as our default options.

4 Big question: Is there any other better line search methods

The big problem we plan to investigate is **"Is there any other better line search methods?"**. The iterative method searches the solution through $x_{k+1} = x_k + \alpha_k p_k,\ p_k = -B_k^{-1}\nabla f_k$ where B_k is a positive definite

matrix ensuring that p_k is a descent direction. The two crucial factors deciding the performance of the algorithm are the step size α_k and the search direction p_k. The line search method determines the search direction p_k first and then calculate the step size by solving the minimization problem of

$$\min_{\alpha_k} f\left(x_k + \alpha_k p_k\right) \tag{1}$$

For the backtracking line search method, (1) is solved approximately by selecting α_k from the largest value in the set of $\{\tau^0, \tau^1, \tau^3 ...\}$ such that both the sufficient decrease condition and curvature condition are both satisfied. There are several observations can be made here: (1) The step size is upper bounded by 1, which may be too conservative as the optimal solution can be larger than 1. (2) The backtracking line search is not intelligent enough in a sense that it simply search α_k by shrinking it in a constant factor of τ. To overcome these potential limitations, we plan to explore other line search methods in this project.

Note that, here we use the original line search method, where all the original step size is 1, to compare new line search methods with the original two line search methods.

4.1 Line search methods

In lectures we studied the inexact line search such as Armijo and Wolfe conditions. There are many other line search methods, and one type of them is exact line search. With exact line search, we expect that the choice of the step size should be better, and one benefit with it may be the less iteration number of the algorithms. We choose two types of exact line search methods in our experiments. One is the section method which does not require the 1^{st} order and 2^{nd} order information and the other one is the exact line search method with interpolation.

4.1.1 Method description: Golden section method

The Golden section method [1, algorithm2.3.1] gives an idea for minimizing a unimodal function over an interval $[a, b]$. The iteration of Golden section method reduce the length of interval $[a_k, b_k]$ to a given precision ϵ_g. One advantages of Golden section method is that it does not require the evaluation of the gradient of the objective function. But the Golden section method is designed for solving a unimodal function which can not be directly applied to the general line search method. When the function is not unimodal in the search direction, it may find a point which is a local minimum but it increases the objective function. Thus We did some modifications to apply the Golden section method as a line search method in our algorithms, which is shown in algorithm 2 in appendix. Our modification makes the line search find a local minimum smaller than the previous update. Then the Golden section method for our line search is as follows. We set the default precision be $\epsilon_g = 10^{-3}$ in the implementation.

4.1.2 Method description: quadratic interpolation

The above golden section search does not use first order information to do the line search. In addition to the gradient free line search method, we may want to improve our inexact backtracking method by utilizing more information. Then, a quadratic interpolation algorithm 3 can come to help. This method is similar to the Newton's method, in a sense that it also builds a local quadratic model to find the minimum along the given direction. Using previous step's function value f_{k-1} and gradient g'_{k-1} and this step's

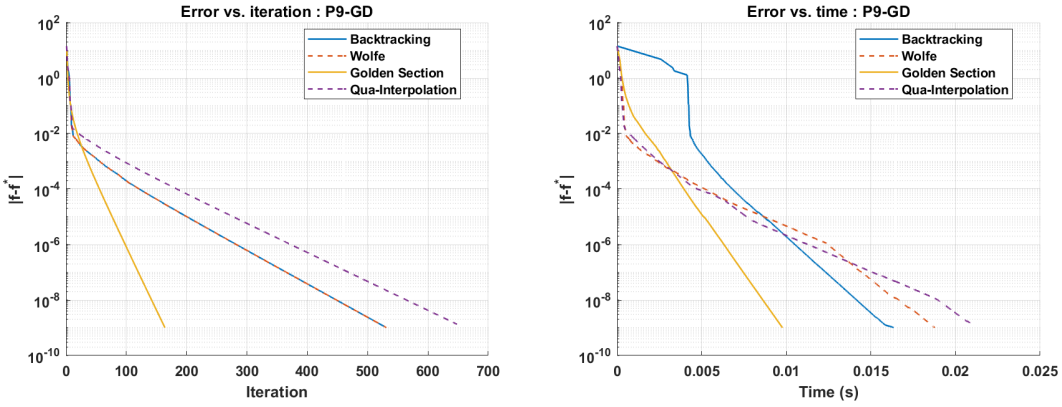

Figure 8: The improvement of Golden section line search on problem 9 using GD.

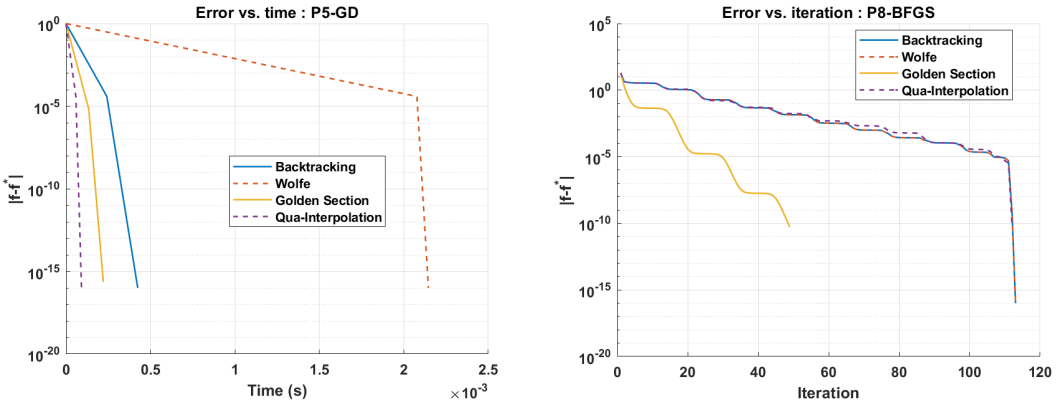

Figure 9: The improvement of Golden section line search on problem 5 using GD and problem 8 using BFGS.

function value f_k, we can do a quadratic interpolation to fit the local quadratic model, and find the local minimum. With this "accurate" finding of step size, we hope a better performance can be found than a simple backtracking. To ensure convergence similar to backtracking method, we set a region γ_1, γ_2 to guarantee the sufficient difference between our fitted local minimum and right hand side bound α. These parameters can also be regarded as the analog of the backtracking's τ.

4.2 Performance comparison

4.2.1 Performance of Golden section line search

Testing our Golden section line search method on different problems. We see that the performance differs on different problems. Our discussion is as follows.

Improvements on performance: The Golden Section method is an exact line search method, and thus we expect it has a faster decrease of the function value. For some problem it did make some improvements. For example, for problem 9 as shown in figure 8. We can see that both number of iteration and running time decreased.

Similar improvements also appears when applying to other problems. For example, in figure 9 we can also see the decrease of the iterations on problem 5 using GD and problem 8 using BFGS.

Discussion on failure of Golden section method: In our experiments, we found that there are cases

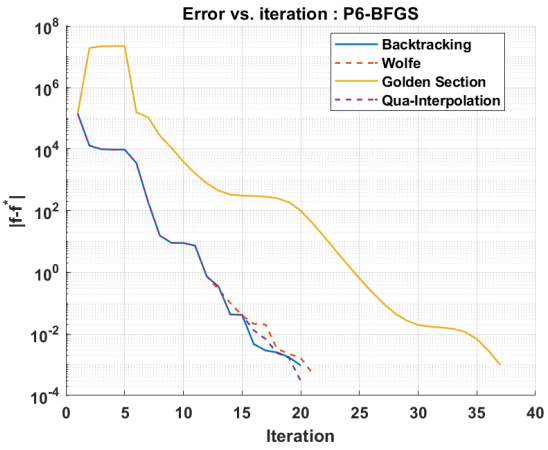

Figure 10: The situation that our implemented Golden section method fails on problem 6 using BFGS.

that our implementation of Golden section line search method fails. For example, on problem 6, it failed to decrease the function value which is illustrated in figure 10. Our explanation is that in our implemented Golden section line search method, we did not check the sufficient decrease on the last update. The termination of Golden section is check the length of the region where has a local minimum. But if the function value still changes dramatically within a small region whose length is less than provided precision ϵ_g, our method does not guarantee the successful decrease of the function value. And problem is possibly under this situation. Thus the success of our Golden section line search requires the objective function to have some degree of Lipschitz continuity.

Other disadvantages: One direct drawback using exact line search such as Golden section method discussed above is the computation time. Though the calculation of gradient is not required for Golden section method to determine the step size, it may take more iterations and time to compute a exact minimum. Another drawback is that with exact minimize the objective in one direction in each iteration does not ensure the fast convergence of the algorithm, because the minimizer in one direction may move the algorithm to a bad location and affects the overall performance.

4.2.2 Performance of Polynomial Interpolation Backtracking

Advantages of the method: When local is true for quadratic or at least convex assumptions in the searching direction, our interpolation will do great on finding the local minimum, in terms of iterations and sub-iterations. And when line search region is small enough, our assumptions mentioned above may always true.

Disadvantages of the method: Local assumptions are too strong to meet, which may lead to a performance worse than simple backtracking.

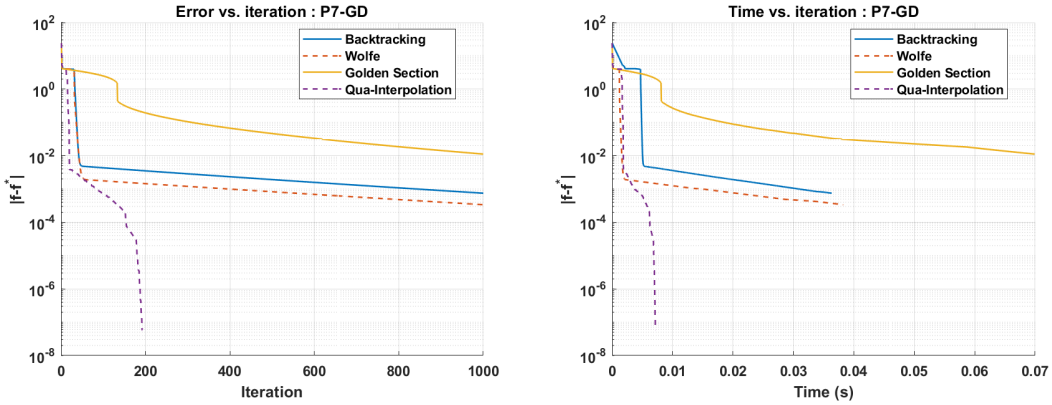

Figure 11: Quadratic interpolation's advantages on P7

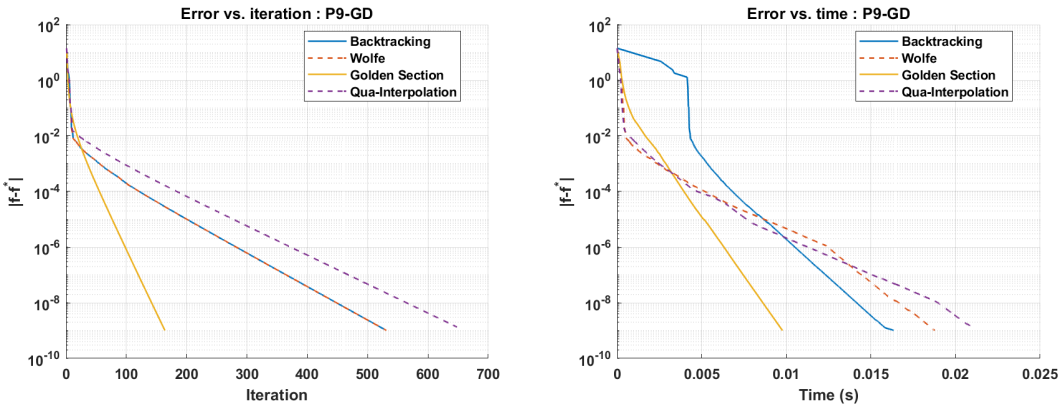

Figure 12: Quadratic interpolation's disadvantages on P9

For P7: Rosebrock with GD can be a good example for the advantages. Plotting the function, you will find smoothness and local convexity near the local minimum. Meanwhile, this is not a very easy global quadratic situation where simple backtracking can also do a great job. These nice properties guarantee quadratic interpolation backtracking's good performance over other line search methods.

For P9: However, when we comes to problem 9, we can find a malfunction of quadratic interpolation backtracking. The only reason we can figure out is the complicated landscape this problem has, where there can hardly be local quadratic.

4.2.3 More extensions:

There are many other exact and inexact line search methods with amazing results and practical use. Some exact methods, like cubic interpolation, quadratic-cubic interpolation, e.t.c.; and tons of inexact methods can be further explore. Also changing the initial step is a good point to seek an improvement, like using quadratic interpolation to be the initial step. Our golden section method and polynomial backtracking method can be the most naive ones.

5 Summary of Experience

Would you declare any of your algorithms the winner? No, every algorithm has its own disadvantages and advantages. From the aspects of convergence speed, computational cost, implementation, parameters' tuning, no always winners. The GD is easy to implement, but the performance is usually worse than other methods. The Newton's method is not too complicated and usually converges fast. But it requires the Hessian information which is not always available for different problems, and when the scale of the problem is large we need to consider more about the computation cost. The Quasi-Newton methods can be good choice if the Hessian information is unknown, however the coding for Quasi-Newton methods is more difficult than other type of line search algorithms. And different types of Quasi-Newton methods performs can differently. The trust region methods works robust on the problem we tested, but the drawback is that coding trust region methods will be more difficult, and there are lots of parameters to tune for trust region methods.

What method would you recommend to an expert coder? Trust region + SR1 CG can be the most balanced method. With a rather good convergence speed in all the problems, the computational cost can be much lower than the Newton based method. The only disadvantage is the difficulty of implementation.

What method would you recommend to a user who is not an expert in coding or in nonlinear optimization? Modified Newton's method can be a good exercise for someone to start. From Newton's method, one can learn the basic ideas of descent direction in optimization. Also based on Newton's method, it could be easy to starting working on Quasi-Newton method such as BFGS and DFP.

References

[1] Wenyu Sun and Ya-Xiang Yuan. *Optimization theory and methods: nonlinear programming*, volume 1. Springer Science & Business Media, 2006.

[2] Mark S. Gockenbach. Backtracking algorithms.

A The results with initial step size be 1

Performance	Algorithm	Problem											
		1	2	3	4	5	6	7	8	9	10	11	12
Iterations	GD	118	1000	112	1000	2	6	1000	42	530	27	21	147
	GDw	33	1000	39	1000	2	6	1000	42	530	27	17	85
	NM	1	1	1	1	2	5	20	4	6	13	13	96
	NMw	1	1	1	1	2	5	20	4	6	13	13	31
	BFGS	25	56	31	358	3	23	33	112	14	19	10	66
	BFGSw	25	52	31	361	3	19	28	112	12	15	9	25
	DFP	35	1000	32	1000	3	56	1000	28	146	15	15	1000
	DFPw	35	840	32	1000	3	67	835	28	18	10	8	104
	TRNMCG	8	12	11	22	3	5	31	4	7	12	13	49
	TRSR1CG	13	19	37	265	3	23	79	71	20	18	18	65

Table 6: Summary of results: with initial step to be 1

B The algorithms of line search methods

Algorithm 2 Golden section

Input initial step size α, precision ϵ_g, function $\phi(x) = f(\tilde{x} + \alpha d)$
Set initial interval $[a_0, b_0] = [0, \alpha]$
while $b_k - a_k > \epsilon_g$ **do**
  $\lambda_k \leftarrow a_k + 0.382 b_k, \mu_k \leftarrow a_k + 0.618 b_k$
  if $\phi(\lambda_k) > \phi(\mu_k)$ **then**
    if $b_k - \lambda_k \leq \epsilon_g$ **then**
      $\alpha \leftarrow \mu_k$
      Break
    end if
    if $\phi(\lambda_k) < \phi(a_k)$ **then**
      $a_{k+1} \leftarrow \lambda_k, b_{k+1} \leftarrow b_k, \lambda_{k+1} \leftarrow \mu_k$ ▷ adding this case to avoid the increase of $\phi(\alpha)$
      $\mu_{k+1} \leftarrow a_{k+1} + 0.618(b_{k+1} - a_{k+1})$
    else
      $a_{k+1} \leftarrow a_k, b_{k+1} \leftarrow \mu_k, \mu_{k+1} \leftarrow \lambda_k$
      $\lambda_{k+1} \leftarrow a_{k+1} + 0.382(b_{k+1} - a_{k+1})$
    end if
  else
    if $\mu_k - a_k \leq \epsilon_g$ **then**
      $\alpha \leftarrow \lambda_k$
      Break
    else
      $a_{k+1} \leftarrow a_k, b_{k+1} \leftarrow \mu_k, \mu_{k+1} \leftarrow \lambda_k$
      $\lambda_{k+1} \leftarrow a_{k+1} + 0.382(b_{k+1} - a_{k+1})$
    end if
  end if
  $k \leftarrow k + 1$
end while
return α

Algorithm 3 Quadratic interpolation backtracking[2, Backtracking algorithms]

Input initial step size α, function value at previous point f, gradient at previous point$g$, direction d, backtracking lower bound γ_1, backtracking upper bound γ_2, sufficient decreasing constant c.

$x_{new} \leftarrow x + \alpha * d$

$f_{new} \leftarrow f(x_{new})$

while $f_{new} > f + c * \alpha * g^T * d$ **do**

$\alpha_{new} \leftarrow \frac{\alpha^2 \nabla f^T(x) d}{2(f(x)+\alpha \nabla f^T(x) d - f(x+\alpha d))}$

if $\alpha_{new} < \gamma_1 * \alpha$ **then**

$\alpha = \gamma_1 * \alpha$

else if $\alpha_{new} > \gamma_2 * \alpha$ **then**

$\alpha = \gamma_2 * \alpha$

else

$\alpha = \alpha_{new}$

end if

$x_{new} \leftarrow x + \alpha * d$

$f_{new} \leftarrow f(x_{new})$

end while

$g_{new} \leftarrow g(x_{new})$

return $x_{new}, f_{new}, g_{new}, \alpha$