Michael Ito (mbito), Mouhamad Obeid (moeobeid) & Ahmad Shmayssani (ahmadhss)

IOE 511/MATH 562: Project
An investigation of the performance of unconstrained nonlinear optimization algorithms

---

Due: April 23rd, 2023

1 Introduction

This report discusses various optimization algorithms and their applications in solving unconstrained optimization problems. Optimization is an essential tool in many areas of science, engineering, and economics, where the objective is to find the best possible solution. In addition, the choice of optimization method and parameters may also depend on factors such as available resources (e.g., computational power, time), the desired level of accuracy or precision, and the patience level of the user. For example, some optimization methods may require more computational resources or time to converge to the optimal solution, while others may be more efficient but less accurate. Additionally, some users may have a higher tolerance for sub-optimal solutions or may be willing to invest more time or resources to obtain a better solution. Therefore, when choosing an optimization method and parameters, it is important to consider these factors and to strike a balance between the desired level of accuracy, available resources, and patience level.

In this context, we will focus on ten optimization algorithms, classified based on the type of search they employ, either trust region or line search. We will describe their working principles, how they compute the steps, and how they handle non-convexity. Additionally, we will compare the algorithms' performance on different optimization problems using various performance metrics, such as convergence speed, optimality gap and performance profiles.

Furthermore, this report aims to answer the research question, *"What are good choices of the line search parameters?"* We will perform sensitivity analysis for τ, c_1 and c_2 for different line search algorithms and observe how they affect the optimization algorithms' performance. The main goal is that these experiments, would provide insight into selecting the optimal line search parameters for a given optimization problem.

Before diving into the performance analysis of each optimization algorithm, it is important to have a basic understanding of how each algorithm works. Understanding the principles behind the algorithms can help in selecting the right algorithm for a specific optimization problem and in interpreting the results. In general, the aim of the algorithms used in this project is to minimize an Objective Function. The algorithms use different strategies to explore the space of possible solutions and to converge towards the optimal solution. These strategies include the use of gradients, hessians, or approximations of these matrices, as well as line search methods to determine the step size at each iteration. The choice of algorithm

depends on the characteristics of the optimization problem, such as the nature of the objective function, the size of the problem, the presence of constraints, and the desired level of accuracy.

It is important to note that although, they all aim to find the minimum of a function they use different methods and techniques to achieve this goal. For this reason we believe is worth introducing the 10 algorithms that we will be using before moving forward.

1. **Gradient Descent with Backtracking line search**: This is a line search algorithm that uses gradient descent to find the minimum of a function. At each iteration, it computes the gradient of the objective function and moves in a descent direction by a step size determined by a backtracking line search. It is suitable for convex problems but can converge slowly for non-convex functions.

2. **Gradient Descent with Strong Wolfe line search**: This is similar to the first algorithm, but uses a more sophisticated line search called the Wolfe condition, which ensures that the step size satisfies both the sufficient decrease and curvature conditions.

3. **Newton with Backtracking line search**: This is a line search algorithm that uses Newton's method to find the minimum of a function. At each iteration, it computes the Hessian matrix of the objective function and moves in the direction of the negative inverse of the Hessian by a step size determined by a backtracking line search. It is suitable for convex and non-convex. It performs well for convex and strongly-convex problems but can be slow for highly non-convex functions.

4. **Newton with Strong Wolfe line search**: This is similar to the third algorithm, but uses the Wolfe condition for the line search. It can converge faster than the third algorithm but is more computationally expensive because the Wolfe condition requires additional function and gradient evaluations at each iteration to satisfy the curvature condition, which increases the computational cost.

5. **Trust Region Newton with CG Steihaug**: This is a trust region algorithm that uses a conjugate gradient (CG) sub-problem solver to compute the step direction. At each iteration, it computes the Hessian matrix of the objective function and solves a trust region sub-problem using the CG method to determine the step size and direction. It is suitable for large-scale problems and can handle non-convexity.

6. **Trust Region SR1 quasi-Newton with CG Steihaug** TRSR1CG is a trust region algorithm that uses the symmetric rank-one (SR1) update to approximate the Hessian matrix and a conjugate gradient (CG) sub-problem solver to compute the step direction. It is a quasi-Newton method that updates the Hessian approximation using the SR1 update formula. This algorithm is similar to TRNewtonCG, but it uses a different method for updating the Hessian approximation.

7. **BFGS with Backtracking line search**: This is a quasi-Newton algorithm that uses the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula to update the Hessian approximation and a backtrack-

ing line search to determine the step size. It is suitable for convex and non-convex problems and can converge faster than gradient descent.

8. **BFGSW with Strong Wolfe line search**: This is similar to the seventh algorithm, but uses the Wolfe condition for the line search. It can converge faster than the seventh algorithm but is more computationally expensive.

9. **DFP with Backtracking line search**: This is a quasi-Newton algorithm that uses the Davidon-Fletcher-Powell (DFP) formula to update the Hessian approximation and a backtracking line search to determine the step size. It is suitable for convex and non-convex problems and is similar to BFGS only differing with the update for the Hessian matrix.

10. **DFPW with Strong Wolfe line search**: This is similar to the ninth algorithm, but uses the Wolfe condition for the line search.

Note that for all Wolfe variants, we used the **Strong Wolfe Condition**

2 Results

The following table provides default options for a code implementation. While the optimality tolerance and iteration limit are set to 1×10^{-6} and 1×10^{3}, respectively, the remaining parameters have been chosen based on extensive testing to ensure optimal performance of the algorithms. We compare all the algorithms for all the problems, using our default parameters, since our aim is to do fair comparisons and comparisons with purpose.

Parameters	**Default Value**
Tolerance	1×10^{-6}
Maximum Iterations	1×10^{3}
c_1 Line Search	1×10^{-4}
c_2 Line Search	0.9
c_1 Trust Region	0.05
c_2 Trust Region	0.8
Δ_0 for trust region	1
Tolerance CG	1×10^{-4}
Maximum Iterations CG	1×10^{3}
β on Cholesky	1×10^{-6}
r on TRSR1	0.2
τ Backtracking Rate	0.5

Table 1: Default Options on Code

Below we provide tables of the numbers of iterations, function evaluations, gradient evaluations, and CPU seconds required to solve twelve test problems using our ten algorithms. Problems 1-4 are quadratics, problems 5-6 are quartics, problem 7-8 are Rosenbrock functions, problem 9 is a data fitting function, problem 10-11 are exponentials, and problem 12 is a Genhumps function.

#Iters.	GD B	GD W	NW B	NW W	TR NW	TR SR	BFGS B	BFGS W	DFP B	DFP W
1	**105**	**105**	**1**	**1**	19	5	26	26	39	39
2	**1000**	**1000**	**1**	**1**	726	50	56	56	**1000**	**1000**
3	116	116	**1**	**1**	**270**	10	31	31	44	44
4	**1000**	**1000**	**1**	**1**	**1000**	50	352	352	**1000**	**1000**
5	**2**	**2**	**1000**	**1000**	**1000**	50	**3**	**3**	**3**	**3**
6	**6**	**6**	38	38	25	50	24	24	**65**	**65**
7	**1000**	**1000**	**20**	**20**	**1000**	32	33	29	47	417
8	42	42	**4**	**4**	**3**	5	**112**	**112**	108	108
9	530	**554**	**6**	**6**	**6**	12	14	14	21	21
10	12	12	**258**	**258**	140	50	**4**	**4**	**4**	**4**
11	32	32	296	296	**181**	50	**24**	**24**	**24**	**24**
12	130	126	92	90	**1000**	**50**	57	140	**1000**	**1000**

Table 2: # Iterations for 12 problems and 10 algorithms

$\#f$.	GD B	GD W	NW B	NW W	TR NW	TR SR	BFGS B	BFGS W	DFP B	DFP W
1	**105**	**105**	**1**	**1**	19	5	26	26	39	39
2	**1000**	**1000**	**1**	**1**	726	50	56	56	**1000**	**1000**
3	116	116	**1**	**1**	**270**	10	31	31	44	44
4	**1000**	**1000**	**1**	**1**	1000	50	352	352	**1000**	**1000**
5	**2**	**2**	**10483**	**10483**	1000	50	3	3	3	3
6	87	87	38	38	**25**	50	86	86	**126**	**126**
7	**9833**	**9773**	**27**	**27**	1000	32	52	58	66	439
8	461	461	4	4	**3**	5	**1011**	**1011**	613	613
9	**2907**	**3033**	**6**	**6**	**6**	12	22	22	29	29
10	12	12	**258**	**258**	140	50	**4**	**4**	**4**	**4**
11	34	34	**296**	**296**	181	50	**29**	**29**	**29**	**29**
12	245	225	118	106	1000	**50**	81	182	1029	**1047**

Table 3: # Function evaluations for 12 problems and 10 algorithms

#∇f	GD B	GD W	NW B	NW W	TR NW	TR SR	BFGS B	BFGS W	DFP B	DFP W
1	**105**	**105**	**1**	**1**	19	5	26	26	39	39
2	**1000**	**1000**	**1**	**1**	726	50	56	56	**1000**	**1000**
3	**116**	**116**	**1**	**1**	270	10	31	31	44	44
4	**1000**	**1000**	**1**	**1**	**1000**	50	352	352	**1000**	**1000**
5	**2**	**2**	1000	**10483**	1000	50	3	3	3	3
6	**6**	87	38	38	25	50	24	86	65	**126**
7	1000	**9773**	20	**27**	1000	32	33	58	47	439
8	42	461	4	4	**3**	5	112	**1011**	108	613
9	530	**3033**	**6**	**6**	**6**	12	14	22	21	29
10	12	12	**258**	**258**	140	50	**4**	**4**	**4**	**4**
11	32	34	**296**	**296**	181	50	**24**	29	**24**	29
12	130	225	92	106	1000	**50**	57	182	1000	**1047**

Table 4: # Gradient evaluations for 12 problems and 10 algorithms

CPU sec.	GD B	GD W	NW B	NW W	TR NW	TR SR	BFGS B	BFGS W	DFP B	DFP W
1	0.088	**0.11**	**0.002**	**0.002**	0.028	0.012	0.025	0.033	0.035	0.045
2	0.90	1.21	**0.002**	**0.002**	1.14	0.12	0.057	0.074	0.97	**1.29**
3	8.04	10.64	**0.23**	0.26	**32.36**	2.02	3.85	4.60	5.03	5.95
4	68.49	90.88	**0.32**	0.36	119.69	9.70	42.54	51.14	112.97	**136.36**
5	0.0002	**0.0001**	0.23	**0.58**	0.12	0.009	0.0004	0.0006	0.0003	0.0004
6	**0.0002**	0.004	0.006	0.006	0.005	0.01	0.004	0.006	0.07	**0.01**
7	0.06	0.19	0.004	**0.002**	**0.10**	0.004	0.003	0.003	0.004	0.03
8	0.07	0.27	0.02	**0.009**	0.01	0.02	0.22	**0.73**	0.17	0.46
9	0.04	**0.13**	0.002	0.002	**0.0009**	0.002	0.002	0.002	0.002	0.003
10	**0.002**	0.003	0.44	**0.45**	0.09	0.04	0.003	0.003	0.003	0.003
11	**0.007**	0.009	0.52	**0.58**	0.12	0.05	0.012	0.015	0.01	0.014
12	0.01	0.02	0.03	0.03	**0.21**	0.02	**0.008**	0.03	0.13	0.15

Table 5: CPU seconds for 12 problems and 10 algorithms

Discussion of Summary Table: We find that for each of the quadratics, Newton with backtracking and Newton with Wolfe line search converge in one step, and thus achieve the best performance out of all the algorithms on all metrics on the convex quadratic problems. This result is expected and occurs because the Newton method uses a second-order model of the objective that matches the true function. In fact, we find that for most problems excluding problems 10 and 11, Newton with backtracking and Newton with Wolfe line search demonstrate the best performance or close to the best performance for each of the metrics. For problems 10 and 11 however, Newton performs poorly. This behavior may occur

because problems 10 and 11 include exponentials, and an exact second-order model is an insufficient approximation to model the objective.

We find that gradient descent is the worst performing model for quadratics with respect to number of iterations, function evaluations, and gradient evaluations. This result is not surprising since gradient descent uses no second-order information about the function, and relies only on the negative gradient as the search direction at each step. For other problems, gradient descent doesn't achieve the best performance for number of iterations, function evaluations, and gradient evaluations, but does perform well in terms of CPU seconds. In comparison to the other algorithms, which require the exact Hessian or an approximation to the Hessian, Gradient Descent is relatively cheap, with its main cost coming from the computation of the function and gradient at each iteration, so this result is not surprising.

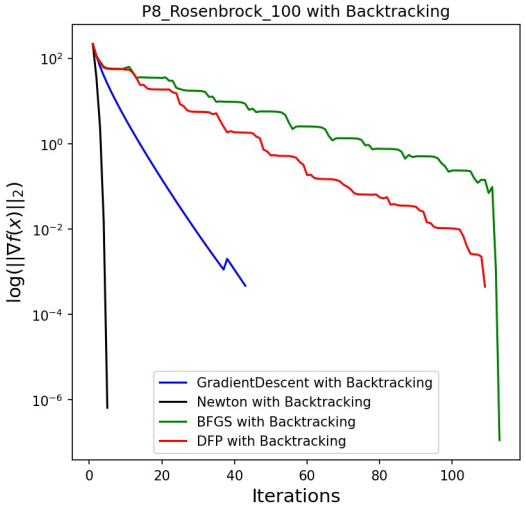
Inspecting the performance for the trust region methods, the trust region newton with CG sub-problem solver method performs poorly on many problems, including some quadratics. This result was surprising since given a large enough starting Δ_0 parameter, we expect trust region newton to converge in a single step similar to Newton with Backtracking or Wofle line search. Thus, in order to improve on performance for the trust region method on the quadratic problems, a more careful selection of the parameter Δ_0 could be chosen. The trust region SR1 algorithm demonstrates good performance for all problems.

BFGS with backtracking and BFGS with Wolfe line search obtained good performance for most problems. Similarly, DFP performed well on most problems. One surprising result was that DFP performed poor on problems 2 and 4, which were convex quadratics. This results was surprising since the performance was opposite of the performance of Newton. One explanation for DFP's poor performance on problems 2 and 4 is that the inverse Hessian approximation is not a good enough approximation to the true inverse Hessian for these particular quadratics, and as a result performed much worse than Newton.

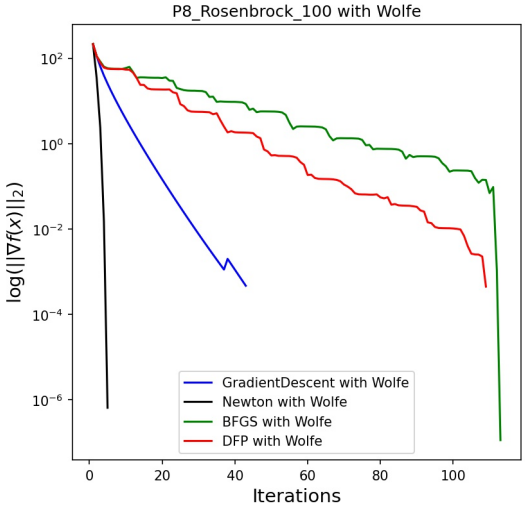
Our expectation prior to the inspection of the results was that Gradient Descent with Wolfe conditions would be our final algorithm of choice. This was our expectation since Gradient Descent is the cheapest algorithm, requiring the least computation and information with only the need to have the gradient. However, despite performing well in CPU seconds, there were various problems such as the quadratics and the Rosenbrock problem where Gradient Descent was the worst performing algorithm with respect to number of iterations, function evaluations, and gradient evaluations. Due to the poor performance on these metrics, Gradient Descent was not the best choice of algorithm. Instead, our algorithm of choice was Newton with Backtracking. Newton with Backtracking obtained good performance on most problems, only obtaining poor performance where exponentials are included. We chose the Backtracking variant because interestingly, for most algorithms the Wolfe line search didn't result in an improvement in performance.

Visualization on Algorithms Performance: After analyzing our ***Summary of Results*** we performed visual comparisons between the different algorithms for all our problems in order to gain some insight.

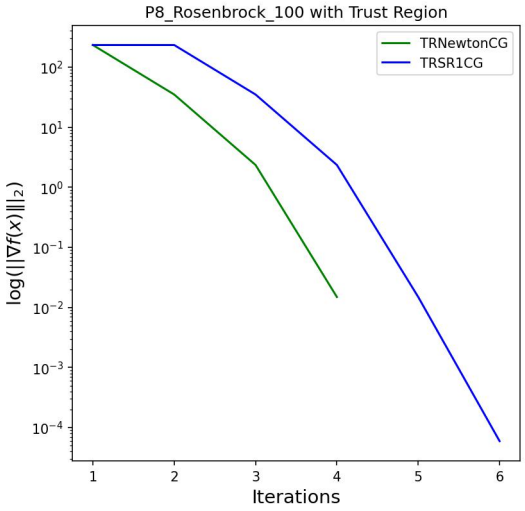
Specifically, we aimed to identify which algorithms' performed best on these problems, as well as any factors that might influence the algorithms performance. To accomplish this, we decided to plot



(a) Problem 8-Armijo Line Search



(b) Problem 8-Wolfe Line Search



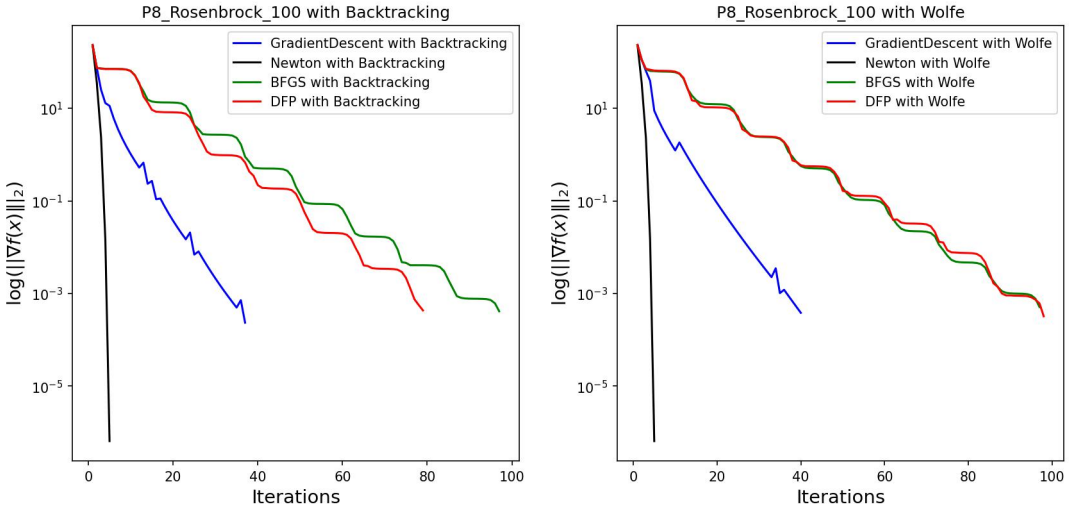
(c) Problem 8 with TR algorithms

Figure 1: $\log(\|\nabla f(x)\|)$ vs Iterations on Problem 8 (Rosenbrock 100)

$\log(\|\nabla f(x)\|_2)$ vs $Iterations$, since we know that as an optimization algorithm converges, the norm of the gradient typically approaches zero. As the algorithm iteratively approaches the minimum, the norm of the gradient (which measures the magnitude of the gradient vector) generally decreases. This is because as the algorithm gets closer to the minimum, the gradient becomes smaller since the slope of the function becomes flatter.

Our visual analysis was divided into three phases, with each phase focusing on Trust Region (TR), Line Search with Backtracking, and Line Search with Strong Wolfe algorithms in separate plots for each problem as shown in Figure 1. This approach can help to provide a clear and detailed understanding of the performance of each algorithm on each problem making more meaningful comparisons. By examining each type of algorithm separately, we observed that typically TR algorithms converge slower than the Backtracking or Wolfe variants for certain problems.

The figures for all the other problems are found on Appendix A, for the purpose of our visual analysis we will mainly focus on Figure 1. We found quite surprising that the Backtracking and Wolfe plots are very similar for most of the problems. As mentioned before here we worked with the Strong Wolfe condition; however to confirm our results we also conducted the analysis with the Weak Wolfe condition and the results were pretty much the same. Then after performing some sensitivity analysis which will be discussed later in Section 3, we realize that if $c_2 = 0.6$ and $c_1 = 0.3$ then the differences start to become quite noticeable between the Backtracking and Wolfe variant and the algorithms converge faster with the Backtracking variant as they only need to satisfy one condition and perform only $f(x)$ computations on each iterations rather than $f(x)$ and $\nabla f(x)$ computations.



(a) Problem 8-Armijo Line Search

(b) Problem 8-Wolfe Line Search

Figure 2: $\log(\|\nabla f(x)\|)$ vs Iterations on Problem 8 (Rosenbrock 100)
(c_2=0.6 and $c_1 = 0.3$)

Additionally, in Figure 2, we observe that both the Backtracking and Wolfe variants of the Newton algorithm converges quadratically on the Rosenbrock function. Notably, all the algorithms converge rapidly and before the maximum iteration limit is reached, satisfying the condition that $\|\nabla f(x)\|_2 \leq \epsilon$. Surprisingly, the gradient descent algorithm performs quite well on this problem, while the quasi-Newton methods perform comparatively worse. Making our decisions only on this set of plots would be very biased. Therefore, we also looked carefully at all the plots found on A we noticed that in general Newton with backtracking performs the best, with its Wolfe variant being the second best. Moreover, it also fails to perform well on Problem 6 this was quite surprising to us but we believed its because due to the high σ that causes numerical instability. It also fails to converge on Problem 10-11 as we previously mentioned, we believe this occurs due to the exponential on the functions. A similar analysis for other algorithms can be seen, BFGS seems to perform relatively well for all the problems except for Problem 8 as shown in Figure 1 where it is the worst performing algorithm. Moreover, DFP does fail to converge in many problems being the worst performing algorithm in the quadratic problems with $\kappa = 1000$ and for the Genhumps

problem. Furthermore, Gradient Descent does poorly in some of the problems typically reaching the iteration limit and failing to converge and it does performs well on the Quartic, Exponential and Genhumps problems. Finally the Trust region problems were quite conflicting. The TRSR1 was barely the best performing algorithm but was consistently performing well and the TR Newton algorithm was quite poor in most of the problems always reaching the iteration limit and taking high CPU time as it can be seen in Table 5

It is important to acknowledge that this analysis may be limited since we are only analyzing a range of problems. However, we believe that this analysis provides us good insights in order to select our algorithm of choice.

Performance Profiles: Our last metric of analysis before selecting our algorithm of choice and answering our big question, was Performance Profiles. When comparing algorithms on a collection of problems, it is important to consider both the fraction of problems solved (i.e., success rate) and the performance metric which in our case was number of iterations. In general, algorithms that perform well on both metrics will be located in the upper left quadrant of the plot, while those that are less effective or efficient will be located in the lower right quadrant. The area under the curve can also be used as a summary statistic for the overall performance of an algorithm, with a larger area indicating better overall performance.

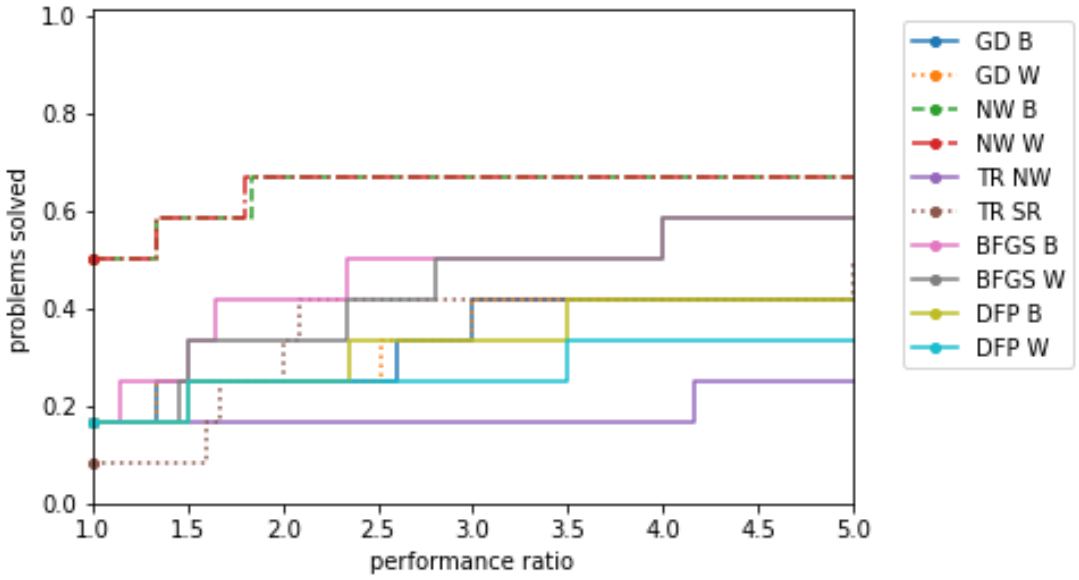


Figure 3: Performance Profiles on Iterations

As shown in the above Figure, we get that Newton with Backtracking and Newton with Wolfe solved 70% of the problems with the best performance measure. Also they are the most robust algorithms, as they achieve the highest success rates across all problems. Then they are followed by BFGS with Wolfe and Backtracking that at a performance ratio of 5 solve around 50% of the problems. The other algorithms have more shallow curves, indicating that they are less robust and may struggle to find good solutions on

a wider range of problems. In terms of efficiency, Newton with Backtracking (NW B) seems to be the most efficient algorithms, as it achieves relatively high success rates with fewer iterations compared to the other algorithms.

Overall, it appears that Newton with Backtracking (NW B) is the best performing algorithm, as it is both robust and efficient across a wide range of problems.

Algorithm of Choice: After evaluating multiple optimization algorithms, the Newton backtracking algorithm was selected as the "Algorithm of Choice" due to its superior performance, excellent convergence properties, and versatility in addressing various optimization problems. To evaluate the algorithm's robustness, a sensitivity analysis was conducted to assess its response to changes in line search parameters. Furthermore, an extension to the Newton Wolfe Algorithm was employed to conduct a more in-depth study of the line search parameters. This analysis will help identify areas for improvement and enable us to evaluate the algorithm's performance under different conditions.

3 Big Question

In optimization, line search parameters are used to determine the step size in each iteration towards the minimum point of the objective function. The choice of these parameters can significantly affect the convergence behavior of the optimization algorithm, and selecting the best values for these parameters is essential to achieving optimal results. To determine the optimal values for the line search parameters, a sensitivity analysis is often performed. A sensitivity analysis involves varying the parameters of interest and observing the resulting changes in the objective function value or other metrics of interest. In this case, the sensitivity analysis involves varying the values of c_1 and τ for the Newton backtracking line search algorithm and varying the values of c_1, c_2, and τ for the Newton Wolfe line search algorithm. Note that for all our analysis we used the same x_0 as provided in the Project Problems file. All tables can be found in Appendix B.

Newton with Backtracking

The parameters that we varied were c_1 and τ for the Newton Backtracking Algorithm for problems 5 and 11 (P5_quartic_1 and Exponential_100). The c_1 range was 5 values between 1e-5 and 0.2 and the τ range was 5 values between 0.7 and 0.01.

Experiment 1: Varying c_1 and τ for problem 5. Refer to Table 6

The best combination of parameters was found to be c_1 = 0.2 and τ = 0.5275, which resulted in a final function value of 0.142. However, it is worth noting that all runs on problem 5 maxed the number of iterations required to reach the given function values. This implies that a higher value of c_1 than is typically used and a nearly standard value of τ may be the most effective combination.

Experiment 2: Varying c_1 and τ for Problem 11. Refer to Table 7

In Experiment 2, the Newton Backtracking algorithm was applied to Problem 11 while varying the

parameters c_1 and τ. Interestingly, all variations of the parameters resulted in the same function value, which suggests that the performance of the algorithm is not sensitive to changes in these specific parameters for this problem. This may be due to the unique structure of Problem 11. Additionally, it is worth noting that the optimal function value was achieved without exceeding the maximum number of iterations and the algorithm stopped at iteration 297 for all parameter variations. This suggests that the algorithm was able to converge efficiently to the optimal solution.

Newton with Strong Wolfe

c_1, c_2 and τ values were varied for the Newton Wolfe algorithm for problem 5 (P5_quartic_1). The c_1 range was 5 values between 1e-5 and 0.2, the c_2 range was 5 values between 0.21 and 0.9, and the τ range was 5 values between 0.21 and 0.9.

Experiment 1: Fixed τ = 0.5, varying c_1 and c_2. Refer to Table 8

In the experiment conducted, it was observed that the best function value was obtained when the value of c_1 was around 0.15, irrespective of the value of c_2. This result suggests that a higher value of c_1 than the conventional value may be more effective in minimizing the function. Additionally, it was found that the maximum number of iterations was reached for all variations, indicating that the optimization process was exhaustive. Therefore, to achieve the optimal function value, one might need to consider using a relatively higher value of c_1 for any value of c_2.

Experiment 2: Fixed c_1=10^{-4}, varying c_2 and τ. Refer to Table 9

In Experiment 2, when c_1 was fixed at 10^{-4} and c_2 and τ were varied, the optimal f value was obtained at c_2=0.3825. However, it was observed that the final f value was sensitive to changes in τ when this value of c_2 was used in combination with the fixed value of c_1. These results suggest that the choice of c_2 is crucial in achieving the optimal function value, while the choice of τ should be made carefully based on the chosen value of c_2.

Experiment 3: Fixed c_2=0.9, varying c_1 and τ for Problem 5. Refer to Table 10

In experiment 5, the effect of varying c_1 and τ for problem X was investigated while keeping c_2 fixed at 0.9. It was observed that the best final function value was achieved when c_1 was set to 0.15. Interestingly, unlike in previous experiments, the final function value was not sensitive to changes in τ when this combination of c_1 and c_2 was used. This suggests that a higher value of c_2 combined with a moderate value of c_1 could be effective in solving this particular problem. However, further experiments may be needed to confirm this observation for a wider range of problems.

In conclusion, our experiments have revealed some surprising findings regarding the choice of parameters in optimization algorithms. For example, in Experiment 2 of the Newton Backtracking algorithm for Problem 11, we found that the algorithm's performance was not sensitive to changes in c_1 and τ. Additionally, in Experiment 3 of the Newton Wolfe algorithm for Problem 5, we found that the final function value was not sensitive to changes in τ when a combination of a moderate value of c_1 and a higher value of c_2 was used. These results suggest that the conventional parameter values used in optimization

algorithms may not always be the most effective.

However, we acknowledge that these results are specific to the algorithms and problems that were evaluated in our experiments. Therefore, to generalize our findings, we need to evaluate them on a wider range of optimization algorithms and problems. Further experiments are needed to confirm the effectiveness of the unconventional parameter values that we found in our experiments and to determine if there are other surprising parameter combinations that could be effective in other optimization problems.

In addition, we believe that expanding the scope of the experiments to include different initial values of x_0 could be an interesting avenue for future research. By exploring the effects of different starting points on the optimization algorithm's performance, we can gain a better understanding of how to choose the best initial point for different optimization problems. Overall, our experiments and proposed extensions provide useful insights for researchers and practitioners looking to optimize their algorithms in real-world settings.

4 Code Deliverable

A brief explanation of the different files relating to the software package.

1. **Application on Rosenbrock Code**: Contains the codes relating to the application of our algorithm of choice on the Rosenbrock function.
2. **Original Code**: Contains all the original codes with no modification to specific uses.
3. **Table Code**: Codes related to the generation of the Table: Summary of Results.

5 Conclusion

Realistically, there it is no feasible way to declare an algorithm better than all other algorithms on all problems. If we are only looking at quadratics for example, Newton may be the best. However, if we are in the regime of a large-scale machine learning problem, Newton may not be feasible due to the full computation of the Hessian. We can on the other hand, consider performance profiles that measure the fraction of problems solved and performance metric for the ten algorithms on the 12 problems. Using the performance profile, our best algorithm of choice across the algorithms and problems was Newton with Backtracking. As discussed in section 2, Newton achieves the best performance with respect to the number of total iterations. Moreover, Newton was relatively easy to code with the only "tricky" part being the Cholesky subroutine. Since we are using backtracking, we only need to tune the two parameters τ and c_1.

Our coding experience is relatively moderate with all members having a background in Python. We would characterize the implementation difficulty into two bins: easy and medium. The easier algorithms

to implement were Gradient Descent and Newton's method since computing the direction was straightforward. Once we obtained the direction, we could execute the Armijo or Wolfe subroutines. Trust Region, BFGS, and DFP were more challenging to implement because computing the direction and trust region were not as straightforward. For example, we found the Trust Region methods more challenging to implement because of the CG Steihaug subroutine involves solving a quadratic model.

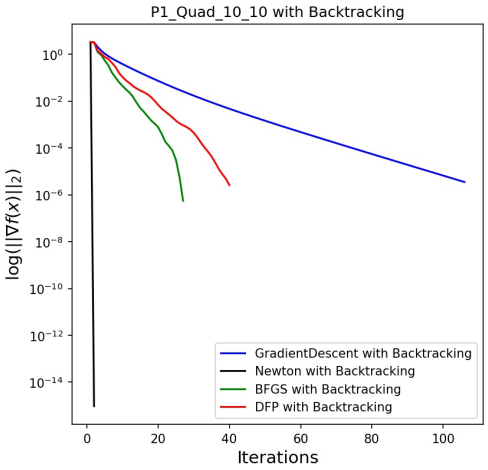
Our recommendation for the particular choice of algorithm is highly dependent on the problem setting. To start, assume we have access to the gradient and the hessian. For a non-expert coder our recommendation is Gradient Descent with Backtracking. If it is infeasible to compute the full gradient, we recommend Stochastic Gradient Descent. Both versions of gradient descent are relatively simple to implement, and require tuning only a few parameters. For an expert coder, Newton with Backtracking is our recommendation since we observed the best performance from Newton with Backtracking. If the hessian is not available, we recommend using BFGS with Backtracking since it obtains a good trade-off between convergence rate and complexity without requiring any second order derivative information.

As an additional comment, we would like to say that after trying to select our algorithm of choice we remembered what Dr. Berahas always used to say in class, **It Depends**. Although we had an algorithm that performed well across our set of problems that does not guarantee that is gonna be the best for another set of problems. For example it is unfair to compare in quadratics Newton to other Algorithms.

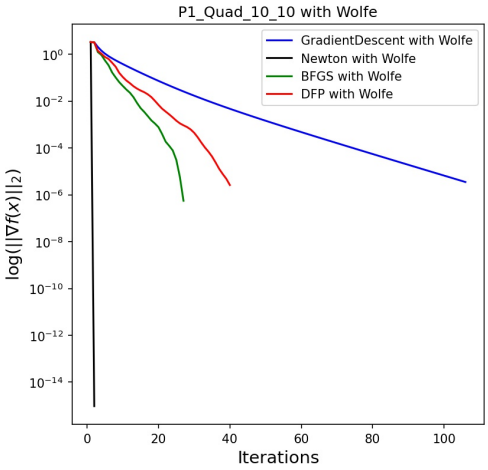
In conclusion, we have found that the choice of optimization algorithm depends on the problem at hand and the user's level of expertise. When selecting an algorithm, one should consider the trade-offs between performance, ease of implementation, and computational resources. Although we have identified the best-performing algorithm in our experiments, it may not be the optimal choice for different problems. Therefore, it is important to carefully evaluate and compare different algorithms for each specific problem before making a decision.

Appendix A Figures of Results

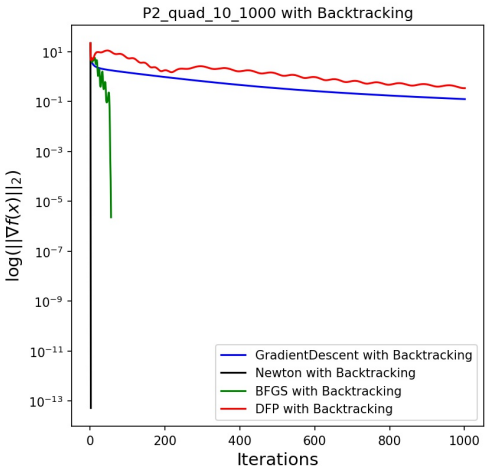
A.1 Figures of Line Search Algorithms



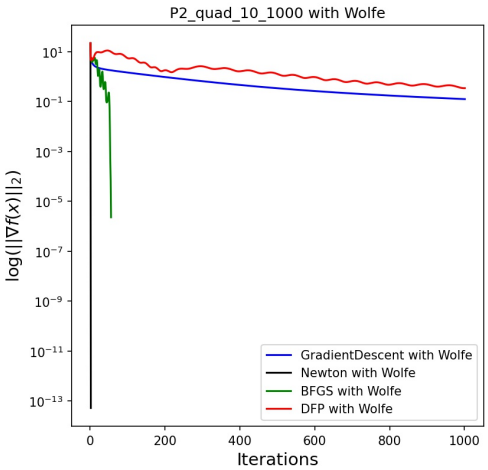
(a) Problem 1-Armijo Line Search



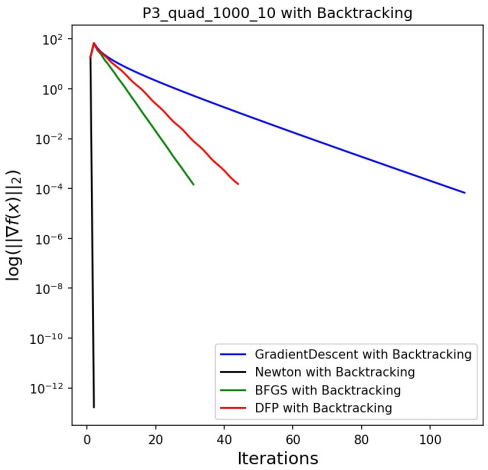
(b) Problem 1-Wolfe Line Search



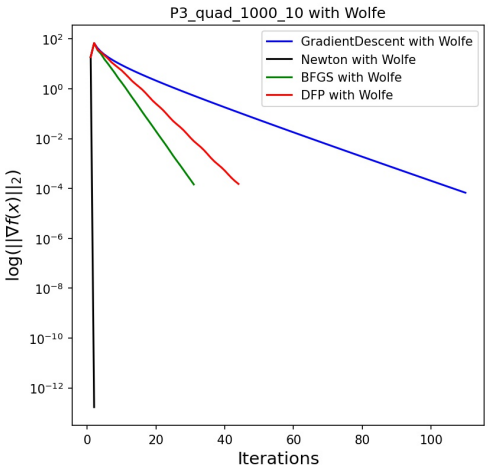
(c) Problem 2-Armijo Line Search



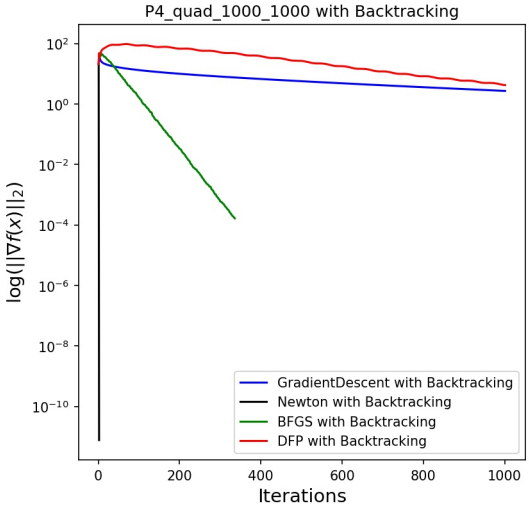
(d) Problem 2-Wolfe Line Search



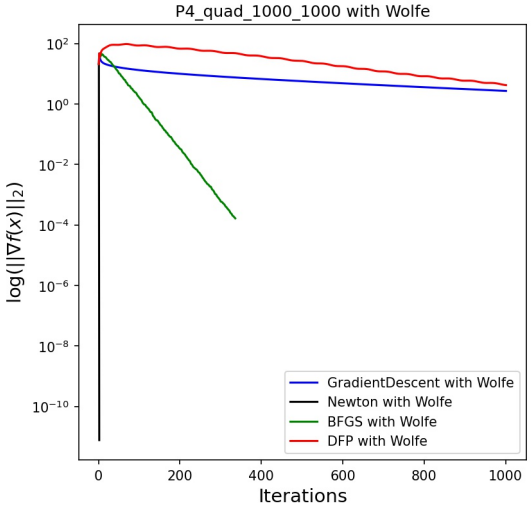
(e) Problem 3-Armijo Line Search



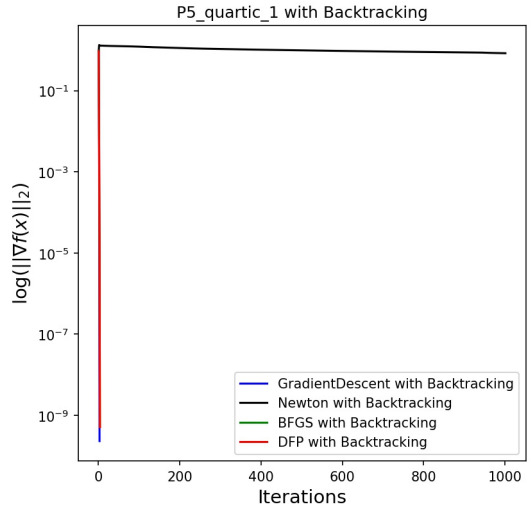
(f) Wolfe Line Search



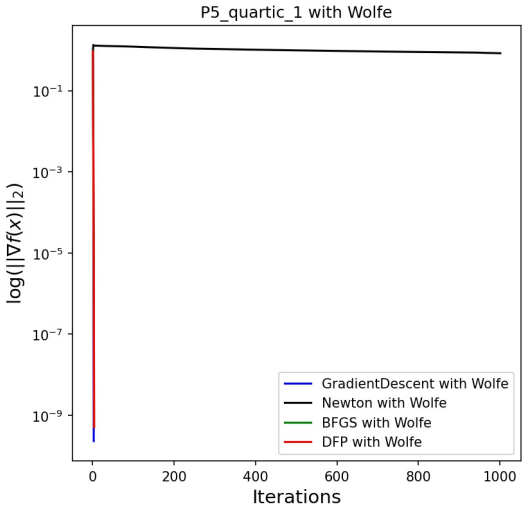
(g) Problem 4-Armijo Line Search



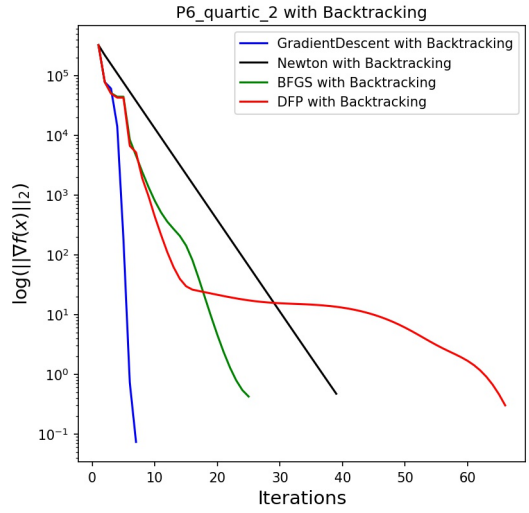
(h) Problem 4-Wolfe Line Search



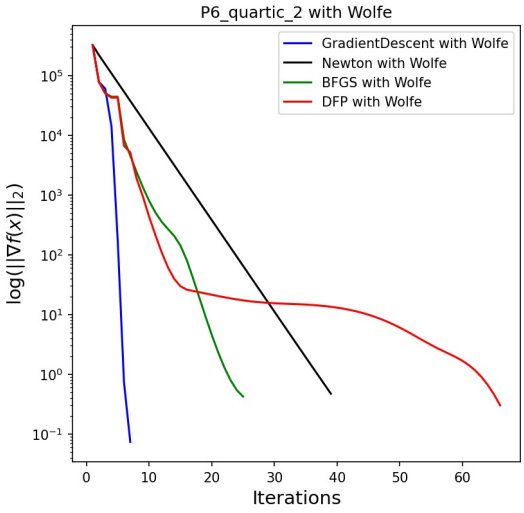
(i) Problem 5-Armijo Line Search



(j) Problem 5-Wolfe Line Search

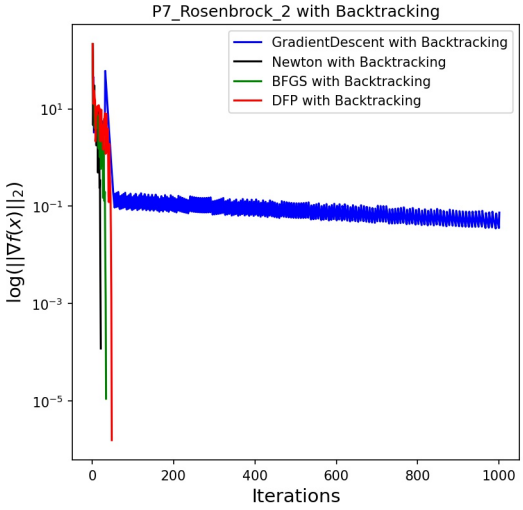


(k) Problem 6-Armijo Line Search

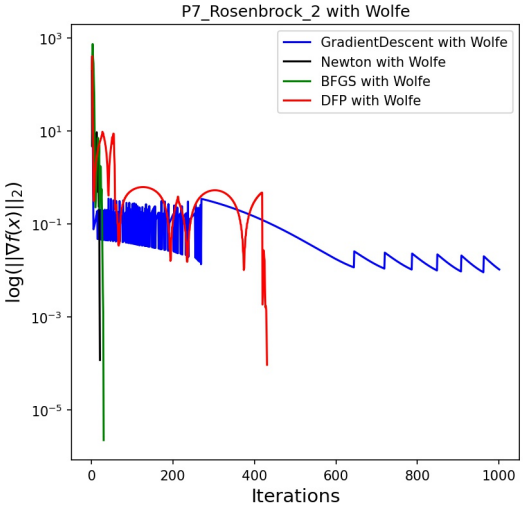


(l) Problem 6-Wolfe Line Search

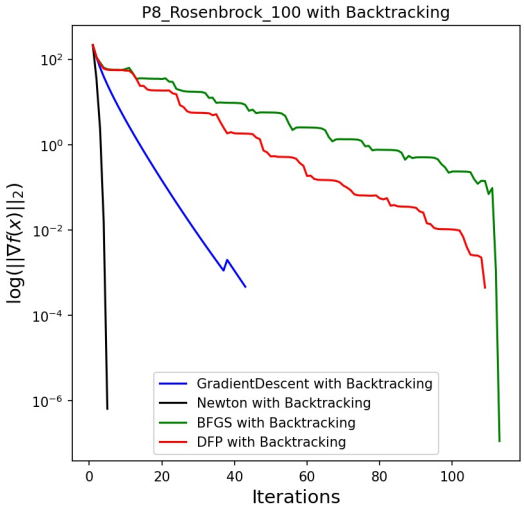
Figure 3: $\log(\|\nabla f(x)\|)$ vs Iterations on Problems 4-5-6 Problems for Line Search Algorithms



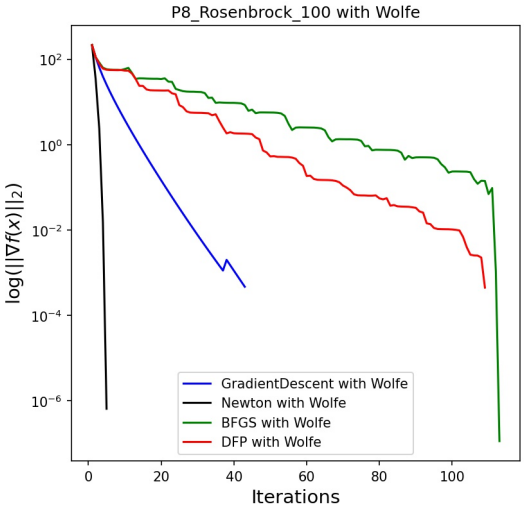
(m) Problem 7-Armijo Line Search



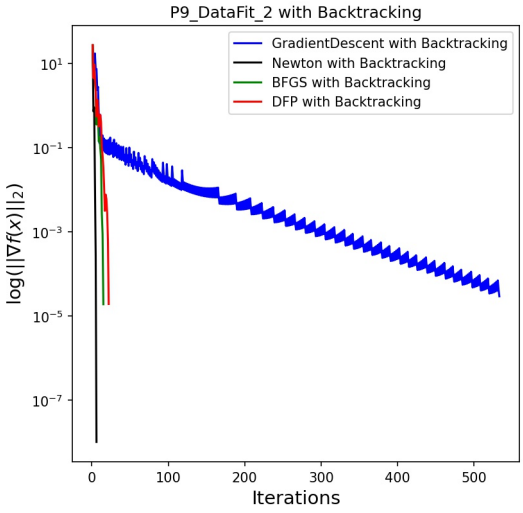
(n) Problem 7-Wolfe Line Search



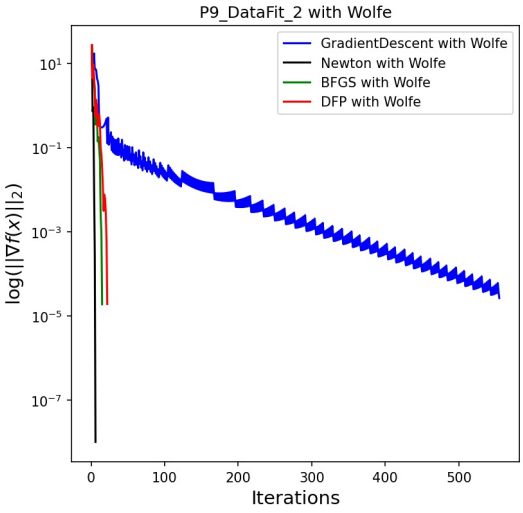
(o) Problem 8-Armijo Line Search



(p) Problem 8-Wolfe Line Search

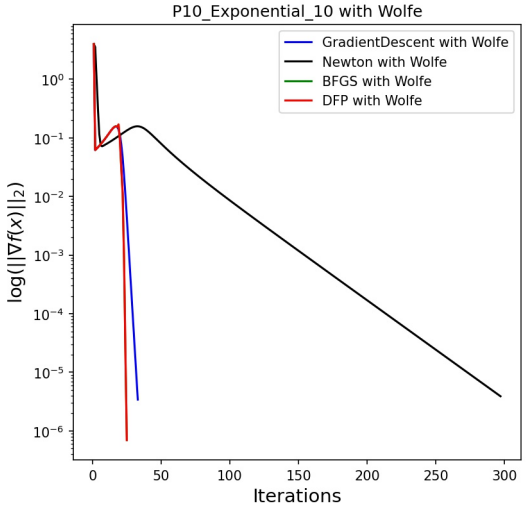


(q) Problem 9-Armijo Line Search

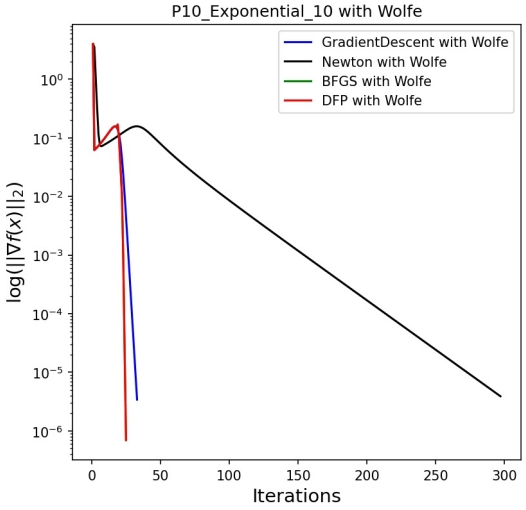


(r) Problem 9-Wolfe Line Search

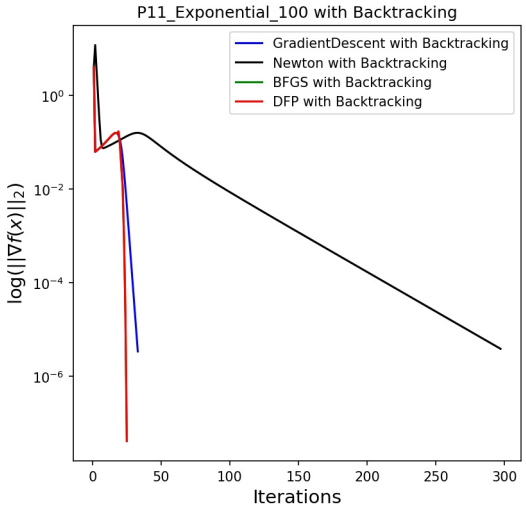
Figure 3: $\log(\|\nabla f(x)\|)$ vs Iterations on Problems 7-8-9 Problems for Line Search Algorithms



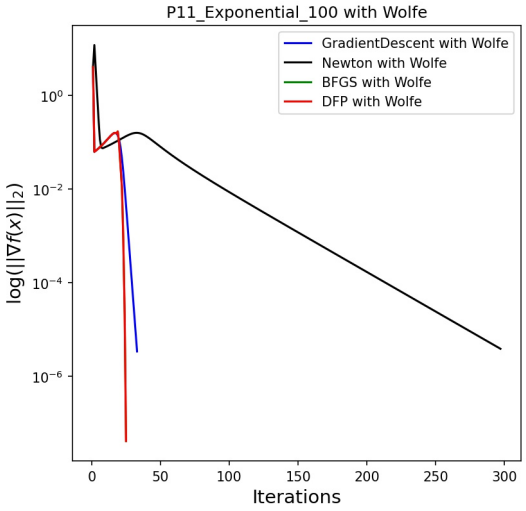
(s) Problem 10-Armijo Line Search



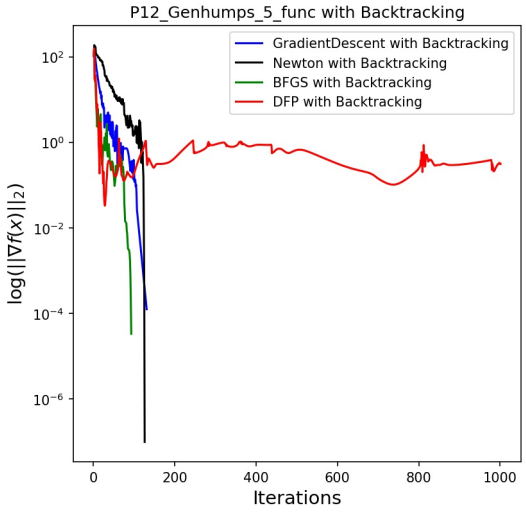
(t) Problem 10-Wolfe Line Search



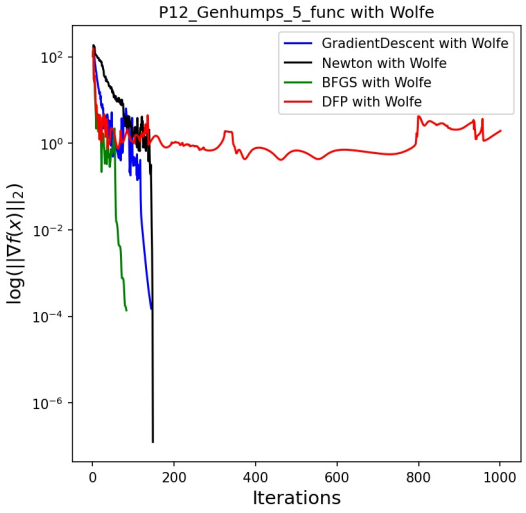
(u) Problem 11-Armijo Line Search



(v) Problem 11-Wolfe Line Search



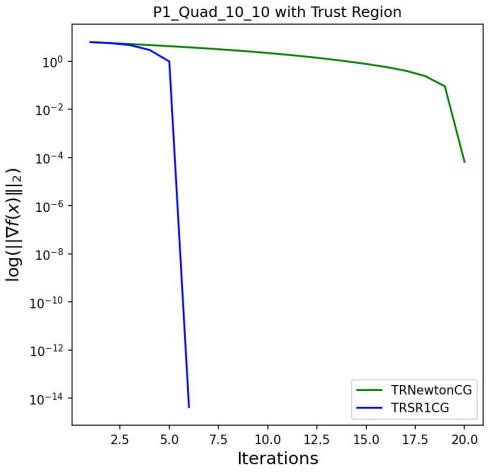
(w) Problem 12-Armijo Line Search



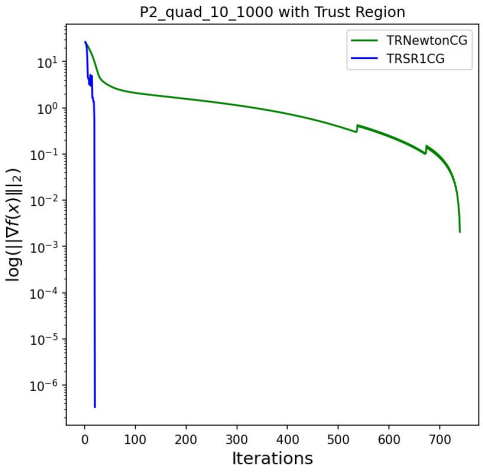
(x) Problem 12-Wolfe Line Search

Figure 3: $\log(\|\nabla f(x)\|)$ vs Iterations on Problems 10-11-12 Problems for Line Search Algorithms

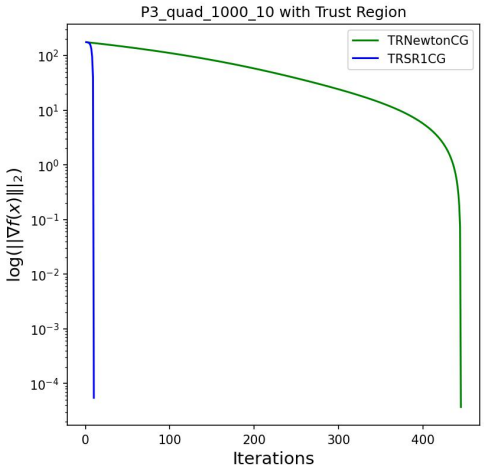
A.2 Figures for Trust Region Algorithms



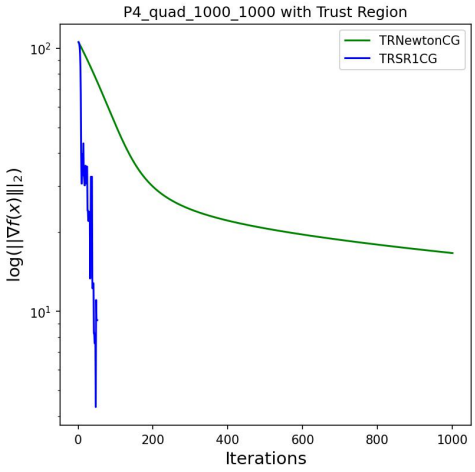
(a) Problem 1 with TR algorithms



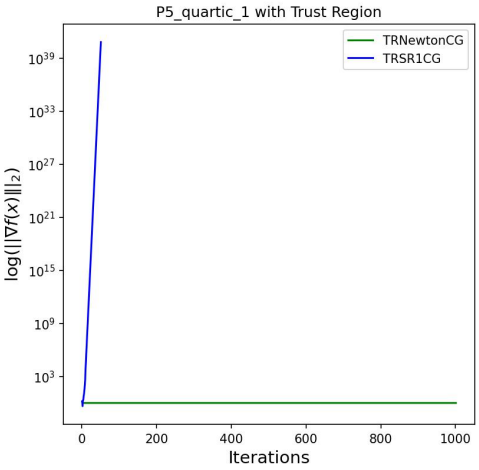
(b) Problem 2 with TR algorithms



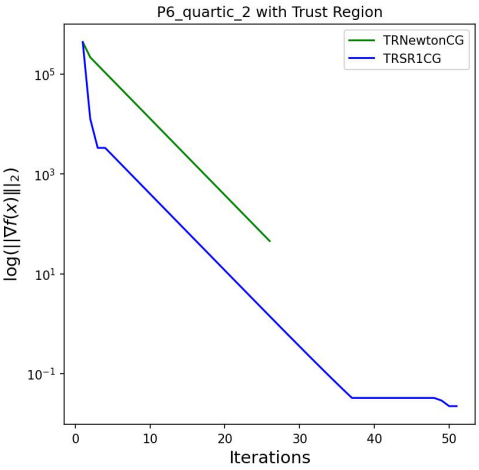
(c) Problem 3 with TR algorithms



(d) Problem 4 with TR algorithms

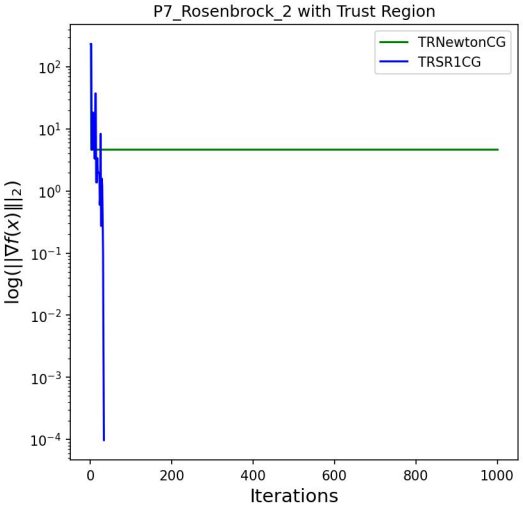


(e) Problem 5 with TR algorithms

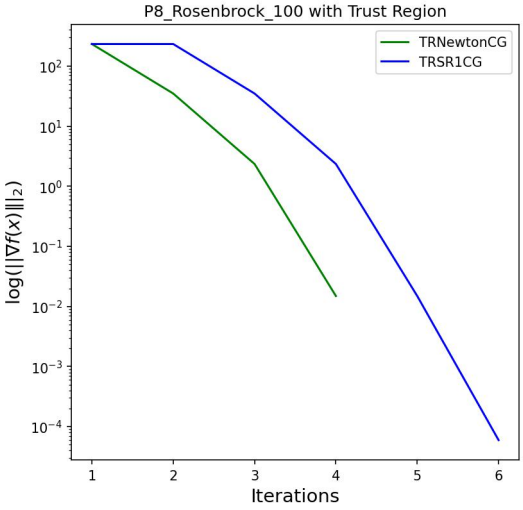


(f) Problem 6 with TR algorithms

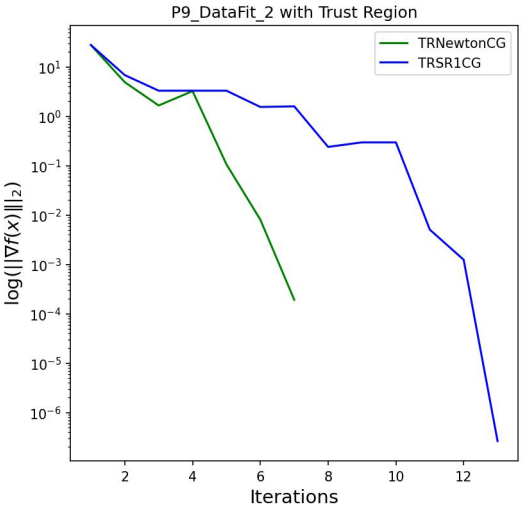
Figure 4: $\log(\|\nabla f(x)\|)$ vs Iterations on All Problems for TR Algorithms



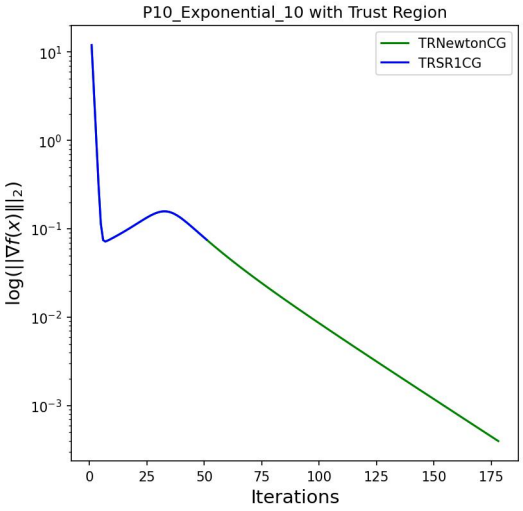
(g) Problem 7 with TR algorithms



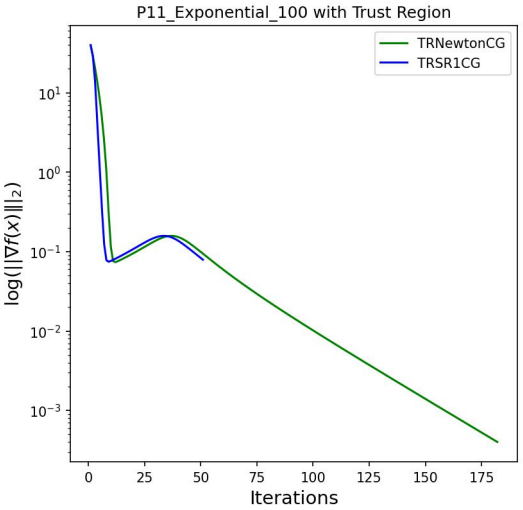
(h) Problem 8 with TR algorithms



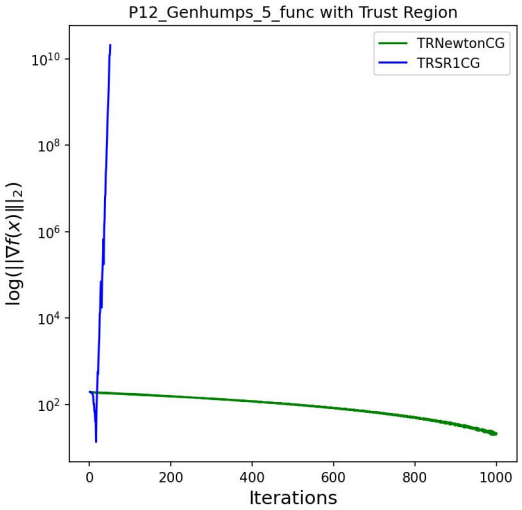
(i) Problem 9 with TR algorithms



(j) Problem 10 with TR algorithms



(k) Problem 11 with TR algorithms



(l) Problem 12 with TR algorithms

Figure 4: $\log(\|\nabla f(x)\|)$ vs Iterations on All Problems for TR Algorithms

Appendix B Tables for Big Question

Trial	c_1	τ	Final f Value	Iterations
0	0.000010	0.7000	0.399396	1000
1	0.000010	0.5275	0.374145	1000
2	0.000010	0.3550	0.298512	1000
3	0.000010	0.1825	0.357169	1000
4	0.000010	0.0100	0.655719	1000
5	0.050008	0.7000	0.398755	1000
6	0.050008	0.5275	0.366782	1000
7	0.050008	0.3550	0.302157	1000
8	0.050008	0.1825	0.355047	1000
9	0.050008	0.0100	0.655738	1000
10	0.100005	0.7000	0.397758	1000
11	0.100005	0.5275	0.364154	1000
12	0.100005	0.3550	0.300675	1000
13	0.100005	0.1825	0.353303	1000
14	0.100005	0.0100	0.655750	1000
15	0.150003	0.7000	0.396090	1000
16	0.150003	0.5275	0.359269	1000
17	0.150003	0.3550	0.303147	1000
18	0.150003	0.1825	0.354349	1000
19	0.150003	0.0100	0.655742	1000
20	0.200000	0.7000	0.394552	1000
21	0.200000	0.5275	0.141899	1000
22	0.200000	0.3550	0.304300	1000
23	0.200000	0.1825	0.354703	1000
24	0.200000	0.0100	0.655770	1000

Table 6: Newton Backtracking Experiment 1 on Problem 5

Trials	c_1	τ	Final f Value	Iterations
0	0.000010	0.7000	-0.205573	297
1	0.000010	0.5275	-0.205573	297
2	0.000010	0.3550	-0.205573	297
3	0.000010	0.1825	-0.205573	297
4	0.000010	0.0100	-0.205573	297
5	0.050008	0.7000	-0.205573	297
6	0.050008	0.5275	-0.205573	297
7	0.050008	0.3550	-0.205573	297
8	0.050008	0.1825	-0.205573	297
9	0.050008	0.0100	-0.205573	297
10	0.100005	0.7000	-0.205573	297
11	0.100005	0.5275	-0.205573	297
12	0.100005	0.3550	-0.205573	297
13	0.100005	0.1825	-0.205573	297
14	0.100005	0.0100	-0.205573	297
15	0.150003	0.7000	-0.205573	297
16	0.150003	0.5275	-0.205573	297
17	0.150003	0.3550	-0.205573	297
18	0.150003	0.1825	-0.205573	297
19	0.150003	0.0100	-0.205573	297
20	0.200000	0.7000	-0.205573	297
21	0.200000	0.5275	-0.205573	297
22	0.200000	0.3550	-0.205573	297
23	0.200000	0.1825	-0.205573	297
24	0.200000	0.0100	-0.205573	297

Table 7: Newton Backtracking Experiment 2 on Problem 11

Trials	c_1	c_2	Final f Value	Iterations
0	0.000010	0.2100	0.818531	1000
1	0.000010	0.3825	0.818531	1000
2	0.000010	0.5550	0.818531	1000
3	0.000010	0.7275	0.818531	1000
4	0.000010	0.9000	0.818531	1000
5	0.050008	0.2100	0.605215	1000
6	0.050008	0.3825	0.605215	1000
7	0.050008	0.5550	0.605215	1000
8	0.050008	0.7275	0.605215	1000
9	0.050008	0.9000	0.605215	1000
10	0.100005	0.2100	0.339733	1000
11	0.100005	0.3825	0.339733	1000
12	0.100005	0.5550	0.339733	1000
13	0.100005	0.7275	0.339733	1000
14	0.100005	0.9000	0.339733	1000
15	0.150003	0.2100	0.304347	1000
16	0.150003	0.3825	0.304347	1000
17	0.150003	0.5550	0.304347	1000
18	0.150003	0.7275	0.304347	1000
19	0.150003	0.9000	0.304347	1000
20	0.200000	0.2100	0.403184	1000
21	0.200000	0.3825	0.403184	1000
22	0.200000	0.5550	0.403184	1000
23	0.200000	0.7275	0.403184	1000
24	0.200000	0.9000	0.403184	1000

Table 8: Newton Wolfe Experiment 1 on Problem 5

Trial	c_2	τ	Final f Value	Iterations
0	0.2100	0.7000	0.366491	1000
1	0.2100	0.5275	0.366491	1000
2	0.2100	0.3550	0.366491	1000
3	0.2100	0.1825	0.366491	1000
4	0.2100	0.0100	0.369656	1000
5	0.3825	0.7000	0.275613	1000
6	0.3825	0.5275	0.275613	1000
7	0.3825	0.3550	0.275613	1000
8	0.3825	0.1825	0.275613	1000
9	0.3825	0.0100	0.286308	1000
10	0.5550	0.7000	0.377412	1000
11	0.5550	0.5275	0.377412	1000
12	0.5550	0.3550	0.377412	1000
13	0.5550	0.1825	0.377412	1000
14	0.5550	0.0100	0.382970	1000
15	0.7275	0.7000	0.399427	1000
16	0.7275	0.5275	0.399427	1000
17	0.7275	0.3550	0.399427	1000
18	0.7275	0.1825	0.399427	1000
19	0.7275	0.0100	0.399431	1000
20	0.9000	0.7000	0.402899	1000
21	0.9000	0.5275	0.402899	1000
22	0.9000	0.3550	0.402899	1000
23	0.9000	0.1825	0.402899	1000
24	0.9000	0.0100	0.403294	1000

Table 9: Newton Wolfe Experiment 2 on Problem 5

Trial	c_1	τ	Final f Value	Iterations
0	0.000010	0.7000	0.818531	1000
1	0.000010	0.5275	0.818531	1000
2	0.000010	0.3550	0.818531	1000
3	0.000010	0.1825	0.818531	1000
4	0.000010	0.0100	0.818531	1000
5	0.050008	0.7000	0.605215	1000
6	0.050008	0.5275	0.605215	1000
7	0.050008	0.3550	0.605215	1000
8	0.050008	0.1825	0.605215	1000
9	0.050008	0.0100	0.605039	1000
10	0.100005	0.7000	0.339733	1000
11	0.100005	0.5275	0.339733	1000
12	0.100005	0.3550	0.339733	1000
13	0.100005	0.1825	0.339733	1000
14	0.100005	0.0100	0.339577	1000
15	0.150003	0.7000	0.304347	1000
16	0.150003	0.5275	0.304347	1000
17	0.150003	0.3550	0.304347	1000
18	0.150003	0.1825	0.304347	1000
19	0.150003	0.0100	0.306736	1000
20	0.200000	0.7000	0.403184	1000
21	0.200000	0.5275	0.403184	1000
22	0.200000	0.3550	0.403184	1000
23	0.200000	0.1825	0.403184	1000
24	0.200000	0.0100	0.402907	1000

Table 10: Newton Wolfe Experiment 3 on problem 5