IOE 511 PROJECT REPORT

Team Dynamix
Chenfei Li, Rachit Garg, Vinayak Bassi

Winter 2023

1 Algorithm Comparison

1.1 Algorithm Description

We study the performance of 10 algorithms on 12 problems. It should also be noted that the symbols used in mathematical equations have standard meanings as discussed in lectures.

(a) Gradient descent with backtracking line search is a line search method. It uses the negative of gradient as the search direction and uses the backtracking method to find the step size.

d=-g

$f(x_k + \alpha_k d_k) <= f(x_k) + c_1 \alpha_k \nabla f(x_k)^T d_k$

(b) Gradient descent with Wolfe line search is a line search method. It uses the negative of gradient as the search direction and uses the Wolfe method to find the step size.

d=-g

$f(x_k + \alpha_k d_k) <= f(x_k) + c_1 \alpha_k \nabla f(x_k)^T d_k$

$\nabla f(x_k + \alpha_k d_k)^T d_k >= c_2 \nabla f(x_k)^T d_k$

(c) Modified Newton with backtracking line search is a line search method. It uses Newton's method to find the search direction and uses the backtracking method to find the step size. Because the hessian of x_k might not be invertible, the Modified Newton method adds H_k with a small multiplier of the identity matrix to make it invertible. Step size is computed by the backtracking method.

$d_k = -[\nabla^2 f(x_k) + \eta I]^{-1} \nabla f(x_k)$

(d) Modified Newton with Wolfe line search is a line search method. It uses Newton's method to find the search direction and uses the backtracking method to find the step size. Because the hessian of x_k might not be invertible, the Modified Newton method adds H_k with a small multiplier of the identity matrix to make it invertible. Step size is computed by the Wolfe method.

(e) Trust region Newton with CG subproblem solver is a trust region method. The CG subproblem uses the Newton's method to find the search direction.

min $m_k(d) := f(x_k) + \nabla f(x_k)^T d + 0.5 d^T B_k d$

s.t. $||d|| <= \Delta_k$

(f) Trust region SR1 with CG subproblem solver is a trust region method. The CG subproblem uses the SR1 quasi-Newton method to find the search direction.

(g) BFGS with backtracking line search is a line search method. It uses the BFGS method to approximate inverse hessian and find the search direction. It uses the backtracking method to find the step size.
min $||H - H_k||$
s.t. $H = H^T, Hy_k = s_k$
$d_k = -H_k \nabla f(x_k)$

(h) BFGS with Wolfe line search is a line search method. It uses the BFGS method to approximate inverse hessian and find the search direction. It uses Wolfe method to find the step size.

(i) DFP with backtracking line search is a line search method. It uses the DFP method to approximate hessian, then estimates inverse hessian, and find the search direction. In our code implementation, we update the inverse hessian using the formula directly. It uses the backtracking method to find the step size.
min $||B - B_k||$
s.t. $B = B^T, Bs_k = y_k$
$d_k = -H_k \nabla f(x_k)$

(j) DFP with Wolfe line search is a line search method. It uses the DFP method to approximate hessian, then estimates inverse hessian, and find the search direction. It uses Wolfe method to find the step size.

1.2 Options Values

We set the default values of parameters in options according to Table 1. In problem 1, we set delta_tr = 17 for the two trust region methods, and the rest parameters use default values. In problem 2, we set c_2_ls = 0.5, delta_tr = 609, and the rest parameters use default values. In problem 3, we set c_2_ls = 0.5, delta_tr = 239, and the rest parameters use default values. In problem 4, we set c_2_ls = 0.5, delta_tr = 8537, and the rest parameters use default values. In problems 7 and 12, we set c_2_ls = 0.5, and the rest parameters use default values. In problems 5, 6, 8, 9, 10, 11, we use default values for all parameters.

options name	value
term_tol	1e-6
max_iterations	1e3
constant_step_size	1e-3
alpha_bar	1
alpha_low	0
alpha_high	1000
c_1_ls	1e-4
c_2_ls	0.9
c_ls_w	0.5
tau_ls	0.5
c_1_tr	0.1
c_2_tr	0.9
c_3_tr	1e-6
delta_tr	0.1
max_iterations_CG	1e3
term_tol_CG	1e-6
beta_Newton	1e-6

Table 1: Default Values

1.3 Algorithm Performance

In general, there is no algorithm that works the best in each problem.

In the four quadratic problems (Problems 1 to 4), the two Newton line search methods and trust region with the Newton method converge in one step, if alpha_bar = 1, and delta_tr is larger than $||x_0 - x^*||_2^2$. This is also supported by the theoretical local convergence result. In Newton's method, by Hessian's Lipschitz continuity assumption, we have:

$||\nabla^2 f(x) - \nabla^2 f(y)|| <= L||x - y||$

And the quadratic problems, $\nabla^2 f$ is constant implies L=0 and thus c=0 in convergence result shown below which is taken from the theorem we prove in lectures where c=ML:

$||x_1 - x^*||_2 <= ML||x_o - x^*||_2^2$

This implies, $||x_1 - x^*|| <= 0$ which means convergence in 1 iteration.

Due to only 1 iteration, Newton's methods are the fastest for the first 4 quadratic problems.

It should also be noted for gradient descent methods, the number of iterations for the second problem is larger than the number of iterations for the first problem. Similarly, the number of iterations for the fourth problem is larger than the number of iterations for the third problem. This is because when the condition number is larger, in general, the convergence constant increases and slows down the rate and it takes more steps for the gradient methods to converge.

Let us consider problem 1, since Q is positive definite, the problem is strongly convex which implies,

$f(x_k) - f^* <= ((\kappa - 1)/\kappa)^k (f(x_o) - f^*)$ where κ is condition number

So, for problem 1 $\rho = (\kappa - 1)/\kappa = 0.9$ and for problem 2, $\rho = 0.999$

It implies more number for iterations for problem 2 as compared to problem 1.

We also noted that Problem 4 takes an extremely long CPU time except for Newton Methods (i.e., Newton backtracking, Newton Wolfe, Trust Region Newton CG) as it takes only 1 iteration. The gradient descent line search methods took a large CPU time due to the combined effect of high dimensionality and high condition number. Quasi-Newton methods also took high CPU time because they converge with a super linear rate for strongly convex functions as per theory which is slower than the quadratic convergence rate of Newton methods. Moreover, we know that the Quasi-Newton method requires a higher number of functions and gradient evaluation at each iteration and this can be observed in Table 4. and Table. 5 also.

In the two quartic problems (Problem 5 and Problem 6), the two gradient descent methods converge the fastest. This aligns with our global convergence result of gradient descent proved for non-convex functions.

In the two Rosenbrock Problems, the two Newton line search methods take the smallest number of steps and CPU time to converge. Additionally, when n = 2 in problem 7, all algorithm converges to the global optimal value. However, when n = 100 in problem 8, all algorithm is stuck on a local optimal and does not reach the global optimal. It also aligns with theory because, in our convergence analysis, we never said that we can converge to a global solution, else we always discussed convergence to a local solution from the neighborhood or any point. Since Rosenbrock is a non-convex function, its optimal solution highly depends on the selected starting point.

In the data fit problem (Problem 9), the BFGS and BFGSW algorithm has the smallest number of iterations and CPU time. On the other hand, DFP and DFPW, two other quasi-Newton methods, have a small number of iterations on Problem 9, but their number of iterations and computational time is much larger than BFGS and BFGSW on Problem 12. In problem 12, we set c_2_ls = 0.5 to make the iteration of DFPW smaller. If we use the default value, it takes DFPW 1000 iterations to converge.

In the Genhump problem (Problem 12), we observed the number of iterations is least for BFGS and BFGSW but CPU time is least for GD backtracking which points out that time per iteration is lesser for GD than BFGS/ BFGSW. It aligns with the theory because Quasi-Newton methods require matrix-vector multiplications at each iteration with an additional computational complexity of $O(n^2)$.

Moreover, for a particular same algorithm, sometimes the backtracking line search method has more iterations than that of the Wolfe line search. However, the number of function evaluations and gradient evaluations of the backtracking method is always smaller or equal to that of the Wolfe line search method. This is because the Wolfe line search method needs to satisfy $f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T d_k$, the condition of backtracking, so it is reasonable that the Wolfe line search method takes more function evaluations to find the step size. Also, backtracking does not evaluate the gradient when finding the step size. Wolfe line search method evaluates $\nabla f(x_k + \alpha_k d_k)$ if $f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T d_k$, so it usually has more gradient evaluations than the backtracking method. As a result, with very few exceptions, the running time of the backtracking method is shorter than the Wolfe line search method given the same algorithm.

In general, based on this problem set, we observed that gradient descent line search methods are consistently performing decently on most of the non-convex problems. We also noted that

there is not much difference in final objective values obtained from different algorithms for each problem. This may be because all algorithms converge to the local minimum. For the convex function (problems 1 to 4), the local and global minimum is the same but not for other non-convex problems. Additionally, most of the convergence plots are sub-linear with some exceptions of convergence in 1 step as discussed and quadratic rate with trust region SR1 method. We also noted some non-smoothness(i.e., decrement in steps) in the convergence plots and they were associated with Wolfe line search every time based on observation. Eg. Problem 7 GD Wolfe, Problem 8 DFP Wolfe, and Problem 10 GD Wolfe. But it was nice to observe that all of the algorithms were converging for the problems.

We also generated performance profiles and relative Area Under Curve rankings where the y-axis of the curve depicts the fractions of problems solved against the allowable performance metrics (i.e., number of iterations, function and gradient evaluations, CPU time). More the area under the curve, the better the performance based on the performance profiles. We obtained gradient descent line search methods performing poorly while BFGS, Newton, and Trust Region Newton CG performed decently and Trust Region SR1 CG performed best. But this should not be taken as the sole criterion to select the best algorithm. This will be covered further in the next section.

	Gradient Descent	Gradient DescentW	Newton	NewtonW	TRNewtonCG	TRSR1CG	BFGS	BFGSW	DFP	DFPW
Problem 1:	105	105	1	1	1	10	26	26	39	39
Problem 2	1000	1000	1	1	1	12	55	51	1000	934
Problem 3	111	38	1	1	1	22	31	31	44	44
Problem 4	1000	1000	1	1	1	1000	366	364	1000	1000
Problem 5	2	2	2	2	5	6	3	3	3	3
Problem 6	5	5	38	38	38	28	28	28	18	18
Problem 7	1000	1000	20	20	31	116	33	28	47	150
Problem 8	42	42	4	4	5	76	112	112	108	108
Problem 9	530	530	24	24	40	20	14	14	21	21
Problem 10	27	27	13	13	14	20	19	20	154	100
Problem 11	21	17	13	13	19	28	10	9	10	9
Problem 12	130	85	90	44	94	74	75	29	1000	165

Table 2: Table: Summary of Results - Number of Iterations

	Gradient Descent	Gradient DescentW	Newton	NewtonW	TRNewtonCG	TRSR1CG	BFGS	BFGSW	DFP	DFPW
Problem 1	0.5128	0.6510	0.0063	0.0080	0.0070	0.0545	0.1353	0.1658	0.1967	0.2464
Problem 2	3.7175	27.954	0.0047	0.0070	0.0060	0.0539	0.2405	0.8288	3.9877	22.250
Problem 3	257.65	417.94	6.5961	8.5723	3.1372	54.724	78.802	100.06	108.15	146.72
Problem 4	696.15	3345.0	1.6462	1.8615	2.4728	787.50	275.48	382.23	761.48	4283.1
Problem 5	0.0000	0.0000	0.0010	0.0010	0.0010	0.0010	0.0000	0.0010	0.0010	0.0010
Problem 6	0.0020	0.0030	0.0119	0.0110	0.0083	0.0060	0.0030	0.0050	0.0050	0.0050
Problem 7	0.2169	0.4171	0.0060	0.0080	0.0110	0.0479	0.0090	0.0120	0.0958	0.0958
Problem 8	0.3267	0.7092	0.0900	0.0743	0.1952	0.8547	1.1640	2.6980	1.4400	2.9561
Problem 9	0.0405	0.1642	0.0848	0.0789	0.0289	0.0120	0.0053	0.0070	0.0073	0.0100
Problem 10	0.0030	0.0040	0.0119	0.0078	0.0040	0.0070	0.0030	0.0060	0.0271	0.0364
Problem 11	0.1217	0.2368	2.2297	2.2582	0.7073	1.7605	1.0067	1.1725	0.5986	0.7571
Problem 12	0.0411	0.1050	0.2564	0.2161	0.1176	0.0619	0.0788	0.0458	0.4393	0.4518

Table 3: Table: Summary of Results - CPU Seconds

	Gradient Descent	Gradient DescentW	Newton	NewtonW	TRNewtonCG	TRSR1CG	BFGS	BFGSW	DFP	DFPW
Problem 1	106	106	2	2	2	11	27	27	40	40
Problem 2	1001	8840	2	2	2	13	56	244	1001	5104
Problem 3	112	201	2	2	2	23	32	32	45	45
Problem 4	1001	5624	2	2	2	1001	367	409	1001	4490
Problem 5	3	3	3	3	6	7	4	4	4	4
Problem 6	73	73	39	39	39	29	97	97	86	86
Problem 7	9834	9883	28	28	32	117	53	100	67	742
Problem 8	462	462	5	5	6	77	1012	1012	614	614
Problem 9	2908	2908	71	71	41	21	23	23	30	30
Problem 10	42	42	31	31	15	21	22	31	164	134
Problem 11	24	27	31	31	20	29	14	19	14	19
Problem 12	246	374	126	191	95	75	115	118	1019	899

Table 4: Table: Summary of Results - Number of Function Evaluations

	Gradient Descent	Gradient DescentW	Newton	NewtonW	TRNewtonCG	TRSR1CG	BFGS	BFGSW	DFP	DFPW
Problem 1	106	106	2	2	2	11	27	27	40	40
Problem 2	1001	9801	2	2	2	13	56	267	1001	5633
Problem 3	112	220	2	2	2	23	32	32	45	45
Problem 4	1001	6202	2	2	2	1001	367	414	1001	4932
Problem 5	3	3	3	3	6	7	4	4	4	4
Problem 6	6	73	39	39	39	29	29	97	19	86
Problem 7	1001	9884	21	28	32	117	34	106	48	817
Problem 8	43	462	5	5	6	77	113	1012	109	614
Problem 9	531	2908	25	71	41	21	15	23	22	30
Problem 10	28	42	14	31	15	21	20	32	155	140
Problem 11	22	28	14	31	20	29	11	20	11	20
Problem 12	131	404	91	217	95	75	76	128	1001	992

Table 5: Table: Summary of Results - Number of Gradient Evaluations

Problem	Gradient Descent	Gradient DescentW	Newton	NewtonW	TRNewtonCG	TRSR1CG	BFGS	BFGSW	DFP	DFPW
1	-24.70	-24.70	-24.70	-24.70	-24.70	-24.70	-24.70	-24.70	-24.70	-24.70
2	-667.13	-673.62	-673.62	-673.62	-673.62	-673.62	-673.62	-673.62	-673.25	-673.62
3	-2014.9	-2014.9	-2014.9	-2014.9	-2014.9	-2014.9	-2014.9	-2014.9	-2014.9	-2014.9
4	-6.86E4	-7.14E4	-7.14E4	7.14E4	-7.14E4	-7.14E4	-71E4	-7.14E4	-7.14E4	-7.14E4
5	1.60E-19	1.60E-19	1.89E-12	1.89E-12	1.02E-17	3.48E-27	1.12E-19	1.12E-19	1.10E-19	1.10E-19
6	0.0010	0.0010	0.0026	0.0026	0.0032	0.0021	0.0013	0.0013	0.0013	0.0013
7	0.75E-03	3.37E-04	8.52E-12	8.52E-12	1.28E-12	4.2E-12	1.87E-13	2.16E-11	2.87E-13	3.81E-11
8	3.98	3.98	3.98	3.98	3.98	3.98	3.98	3.98	3.98	3.98
9	1.01E-09	1.01E-09	1.14E-10	1.14E-10	7.78E-12	7.29E-15	3.8E-12	3.8E-12	3.72E-10	3.72E-10
10	-0.20	-0.20	-0.20	-0.20	-0.20	-0.20	-0.20	-0.20	-0.20	-0.20
11	-0.20	-0.20	-0.20	-0.20	-0.20	-0.20	-0.20	-0.20	-0.20	-0.20
12	3.88E-08	4.24E-08	3.16E-10	1.97E-11	2.54E-09	1.33E-08	2.51E-09	1.05E-08	0.20355	8.53E-08

Table 6: Table: Summary of Results - Final Function Value

1.4 Best Algorithm

The choice of the best algorithm is multi-objective in nature. Our choice is also based on a combination of factors like the minimum number of iterations, minimum time, the minimum number of function and gradient evaluations, convergence speed, and minimum objective value. We observe that Modified Newton with backtracking is performing best overall on the given problems. If we consider the total CPU time of all the problems, then this method takes a minimum. The same is observed for the number of iterations, function, and gradient evaluations. This also points to the effect-causality relation between the two observations. As per the theory, Modified Newton has local convergence results. It converges faster if the starting point is in the neighborhood of the optimal solution but problematic if not. For the given set of problems and starting points, the

final objective function values remain in a similar range for most of the problems. So, they are not higher for Modified Newton with backtracking, and even the least final objective value is obtained with this method. It points out that the property of local convergence is not having an adverse effect on Modified Newton on the given problem set. We are recommending backtracking with it because of the observed results as explained. It can also be aligned with theory as backtracking requires satisfying fewer conditions on step size as compared to Wolfe. This leads to less number of function and gradient evaluations also for backtracking and in turn less CPU time. It should be noted that less CPU time is not having any trade-off with minimum final function value so, this can be safely selected.

The selection of Modified Newton with backtracking as the best choice is having some misalignment with the results from the performance profile and relative AUC rankings but this can be explained. Though Trust Region SR1 CG ranks 1, it does not perform nicely based on the number of iterations, gradient and function evaluations, and CPU time, especially on Problem 4 where it is not up to the mark. Similar, can be observed for other following top ranking algorithms like BFGS, TR Newton CG as per relative AUC rank. But Modified Newton with backtracking strikes a perfect balance between them. It has a relative area under the curve of 0.7 which is also a good value. Moreover, there is not a significant difference in relative AUC for BFGS line search, TR Newton CG, and Newton line search methods. Hence, we propose 'Modified Newton with backtracking' as our best choice algorithm. It is based on heuristics and balancing and may be a little biased due to the inclusion of problem 4.

```
Relative area under the curve ranking:
             TRSR1CG has area 9.378642e+02 and relative area (to maximum) 1.000000e+00
                BFGS has area 7.746125e+02 and relative area (to maximum) 8.259325e-01
               BFGSW has area 7.313008e+02 and relative area (to maximum) 7.797513e-01
          TRNewtonCG has area 7.279692e+02 and relative area (to maximum) 7.761989e-01
             NewtonW has area 6.596700e+02 and relative area (to maximum) 7.033748e-01
              Newton has area 6.580042e+02 and relative area (to maximum) 7.015986e-01
                 DFP has area 4.947525e+02 and relative area (to maximum) 5.275311e-01
                DFPW has area 4.780942e+02 and relative area (to maximum) 5.097691e-01
    GradientDescent has area 3.348325e+02 and relative area (to maximum) 3.570160e-01
   GradientDescentW has area 3.331667e+02 and relative area (to maximum) 3.552398e-01
```

2 Big Question: What is the best line search parameter?

The best line search parameter always depends on the problem and algorithm ideally.

All algorithms use line search methods except TRNewton and TRSR1.

We tune the parameters on a subset of discrete values (refer to Table 8.), and run on the combinations of the parameter values. We have replicated the process for each line search algorithm and thus, obtained the distinct best combination of line search parameters for each line search algorithm. For this process, we have run each algorithm with the parameter grid on only *Rosenbrock*_100 (Problem 8). It is to slightly simplify the process rather than combining various metrics and plots for all the problems. *Rosenbrock*_100 (Problem 8) is a non-convex problem and we can safely use it and assume similar inferences on other problems.

For the backtracking line search method, we tune on alpha_bar, tau, and c_1_ls. alpha_bar determines the initial step size. If it is too large, the line search method might decrease fast in the beginning, but as the number of iterations increases, it might take too many times to get the step size. tau determines the reduction of step size. c_1_ls determines when to stop. If it is too large, we

might take too many iterations to get a satisfied new point that significantly reduces the function value.

For the Wolfe line search method, we tune on alpha_bar, c_1_ls, and c_2_ls. By default, we set c_2_ls to 0.9, a value close to 1. When c_2_ls is too small, the time of finding step size might become large, and the number of function and gradient evaluations could also increase a lot.

After getting conventional resultant metrics like the number of iterations, function and gradient evaluations, CPU time, and final function value; we configured a custom-built metric named 'score' to select the best combination of parameters by balancing all the mentioned conventional resultant metrics in appropriate proportion as below. We are considering the best parametric combination with a maximum score and the worst with a minimum score. We have not considered the final function value in scoring as it was not deviating much.

Score = - [0.25*(Number of Iterations) + 0.25* (CPU Time) + 0.25* (Number of Function Evaluations) + 0.25* (Number of Gradient Evaluations)]

alpha_bar	0.05, 0.1, 1, 10, 100
tau	0.005, 0.02, 0.1, 0.5, 0.9
c_1_ls	10^{-12}, 10^{-8}, 10^{-4}, 0.006, 0.6

Table 7: Parameter Value - Backtracking

alpha_bar	0.001, 0.2, 1, 10, 100
c_1_ls	10^{-8}, 10^{-4}, 0.01, 0.006
c_2_ls	0.02, 0.5, 0.65, 0.9

Table 8: Parameter Value - Wolfe

2.1 Backtracking Line Search

We observed the same final function value across best, worst, or default parameters in all the algorithms. In gradient descent, the best score is obtained after reducing α, τ, c_1 from default values. It can be because as c_1 reduces, the strictness on the selection of α also relaxes. When we increased α, τ from default values, it becomes the worst score which may be because the algorithms surpass or deviate from the optimal solution at some iterations due to large step size. For BFGS, we noted that the number of function evaluations is significantly decreasing with the reduction in α, τ, c_1 from default values which is increasing the score majorly and outweighing other performance metrics. As strictness on the selection of α reduces, it requires fewer conditions to satisfy and fewer functions to evaluate. In DFP, as c_1 reduces to 10^{-8} it is the best parameter set, but as it further reduces to 10^{-12}, it becomes worst. This also records heavy fluctuations in CPU time for best and worst from the default settings. In general, best parameter sets are corresponding to reduced α and τ and vice versa for worst. We also recorded some differences between our custom best score and convergence plots. In DFP, the best score parameters have a poor rate of convergence as compared to the worst score parameters. The rate of convergence can also be included in the scoring metric for future implications where the rate of convergence is also significant based on complex and heavier problems.

	(alpha_bar, tau, c_1_ls)	Final F Value	No. Iterations	Computational Time	No. Function Evaluations	No. Gradient Evaluations	Score
Best	(0.05, 0.02, 1e-08)	3.9866	46	0.1348	93	47	-46.7837
Default	(1, 0.5, 0.0001)	3.9866	42	0.3267	462	43	-136.832
Worst	(100, 0.9, 0.0001)	3.9866	82	8.4491	9002	83	-2294.1123

Table 9: GradientDescent with Backtracking Line Search

	(alpha_bar, tau, c_1_ls)	Final F Value	No. Iterations	Computational Time	No. Function Evaluations	No. Gradient Evaluations	Score
Best	(1, 0.1, 0.006)	3.9866	4	0.0319	5	5	-3.7580
Default	(1, 0.5, 0.0001)	3.9866	4	0.0479	5	5	-3.7620
Worst	(1, 0.005, 0.6)	3.9879	1000	9.0758	2001	1001	-1003.0189

Table 10: Newton with Backtracking Line Search

	(alpha_bar, tau, c_1_ls)	Final F Value	No. Iterations	Computational Time	No. Function Evaluations	No. Gradient Evaluations	Score
Best	(0.1, 0.02, 1e-08)	3.9866	134	0.6444	246	135	-129.1611
Default	(1, 0.5, 0.0001)	3.9866	112	0.4753	1012	113	-309.6188
Worst	(100, 0.9, 0.0001)	3.9866	174	14.7874	13028	175	-3348.1969

Table 11: BFGS with Backtracking Line Search

	(alpha_bar, tau, c_1_ls)	Final F Value	No. Iterations	Computational Time	No. Function Evaluations	No. Gradient Evaluations	Score
Best	(1, 0.005, 1e-08)	3.9866	118	0.2754	231	119	-117.3189
Default	(1, 0.5, 0.0001)	3.9866	108	0.8728	614	109	-208.2182
Worst	(100, 0.9, 1e-12)	3.9866	164	7.7544	9644	165	-2495.4386

Table 12: DFP with Backtracking Line Search

2.2 Wolfe Line Search

We observed the same final function value across best, worst, or default parameters in all the algorithms. In gradient descent, the best score is obtained after reducing α, increasing c_1, and decreasing c_2 from default values. It may be because as c_1 increases and c_2 reduces, the strictness on the selection of α increases and may help to select better α. Else it could also be dominated by the reduced value of α which avoids the issue of passing the optimal function value in some iterations. When we increased α with some variations in c_1 and c_2 but not making it strict, it becomes the worst score which may be because the algorithms surpass or deviate from the optimal solution at some iterations due to large step-size. But this effect of α cannot be generalized because we observed the ill-effect of reduced step size in the Newton method. In Newton, as we decreased α by

keeping c_1 and c_2 constant, it increased the number of iterations, function and gradient evaluations, and CPU time. This again flips for BFGS. We noted that all the conventional metrics decreased with a significant decrease in α. In general, we do not record any specific guarantee of improving solution on monotonically decreasing α. But we observed the best solution after making the Wolfe conditions more strict i.e., increasing c_1 and reducing c_2 with respect to a default value and this was also associated with reduced α for all algorithms except Newton method. Unlike backtracking, We do not see much difference between our custom best score and convergence plots for Wolfe. But in BFGS and DFP, the worst score parameters have a better rate of convergence as compared to default parameters. The rate of convergence can also be included in the score metric for future implications as previously suggested.

	(alpha_bar, c_1_ls, c_2_ls)	Final F Value	No. Iterations	Computational Time	No. Function Evaluations	No. Gradient Evaluations	Score
Best	(0.05, 0.006, 0.02)	3.9866	36	0.1127	239	239	-128.7782
Default	(1, 0.0001, 0.9)	3.9866	42	0.2161	462	462	-241.8040
Worst	(100, 0.006, 0.5)	3.9866	35	0.3122	616	616	-317.0781

Table 13: GradientDescent with Wolfe Line Search

	(alpha_bar, c_1_ls, c_2_ls)	Final F Value	No. Iterations	Computational Time	No. Function Evaluations	No. Gradient Evaluations	Score
Best	(1, 0.01, 0.9)	3.9866	4	0.0338	5	5	-3.7584
Default	(1, 0.0001, 0.9)	3.9866	4	0.0402	5	5	-3.7601
Worst	(0.1, 0.0001, 0.9)	3.9866	124	1.2623	134	135	-98.8156

Table 14: Newton with Wolfe Line Search

	(alpha_bar, c_1_ls, c_2_ls)	Final F Value	No. Iterations	Computational Time	No. Function Evaluations	No. Gradient Evaluations	Score
Best	(0.05, 0.0001, 0.9)	3.9866	111	0.5055	557	558	-306.8764
Default	(1, 0.0001, 0.9)	3.9866	112	0.92334	1012	1012	-534.4808
Worst	(100, 0.006, 0.5)	3.9866	116	4.8890	1793	1793	-926.9722

Table 15: BFGS with Wolfe Line Search

	(alpha_bar, c_1_ls, c_2_ls)	Final F Value	No. Iterations	Computational Time	No. Function Evaluations	No. Gradient Evaluations	Score
Best	(0.05, 0.01, 0.9)	3.9866	112	0.5164	279	283	-168.8791
Default	(1, 0.0001, 0.9)	3.9866	108	1.9798	614	614	-334.7450
Worst	(100, 0.006, 0.02)	3.9866	98	4.8323	1236	1236	-643.9581

Table 16: DFP with Wolfe Line Search

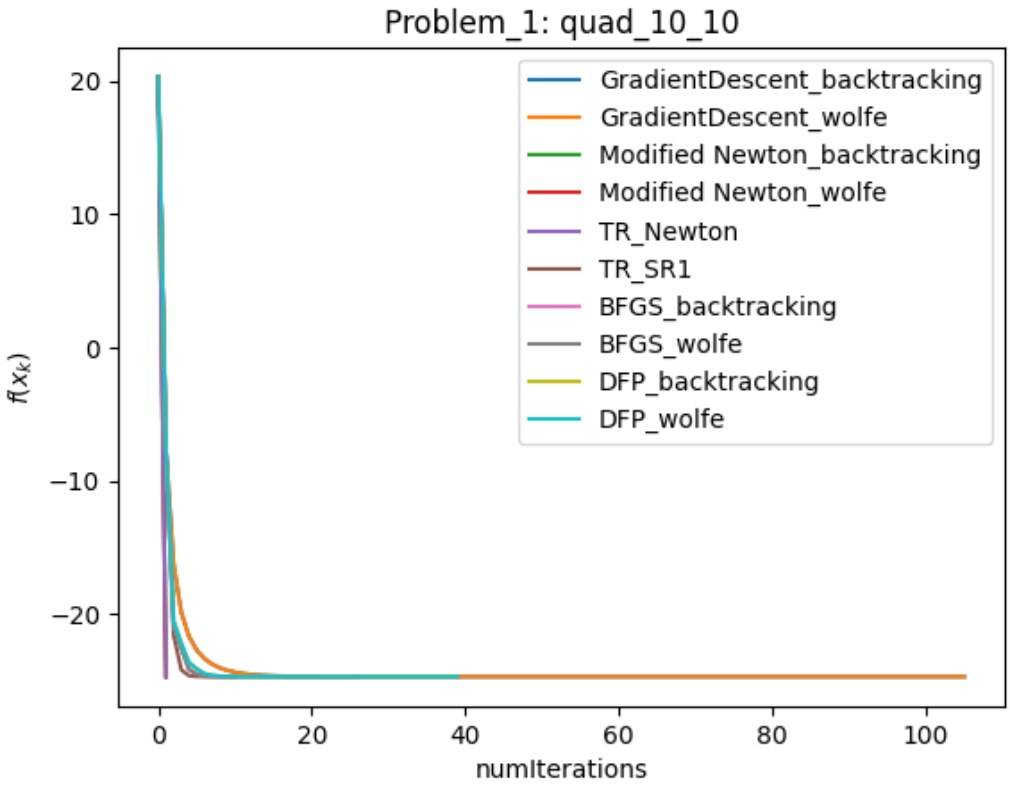
3 Conclusion

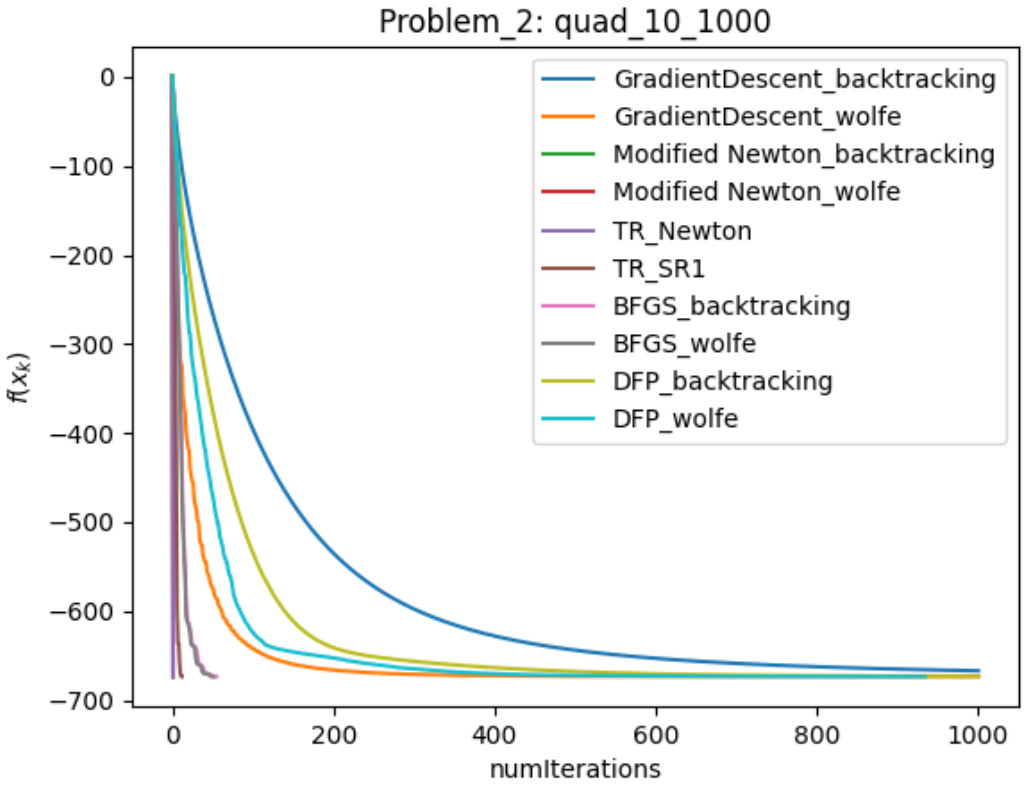
We should not declare any algorithm as a winner because it depends on various factors like performance analysis on a large set of diverse problems, ease of implementation, final object value, etc. These factors can have different weights based on the application. We observed that Wolfe line search methods incorporate a higher number of parameters which makes the parameter-tuning task a little cumbersome. The same is observed with both the trust region methods also. On the other hand, backtracking line search methods involves relatively less parameter tuning. We felt mixed reviews on the implementation of each algorithm. Gradient descent backtracking was most easy to implement as it involves the least coding and the least number of parameters to select default values or tune. Gradient descent Wolfe is also similar with a few more parameters that increase work. While implementing Modified Newton with backtracking, we incurred significant coding efforts as it required calculating hessian also and estimating η based on Cholesky decomposition to ensure it is always invertible. And the number of parameters to select default values further increased with selecting α based on the Wolfe line search. We required similar coding efforts for trust region Newton CG as well as trust region SR1. They both required to deal with δ_0 as well which we were not much comfortable with as line search methods are more popular. The other Quasi-Newton methods were not much challenging to implement because they do not require the calculation of hessian. The only effort was to code a large complex expression of inverse hessian approximation accurately as it was prone to silly mistakes.

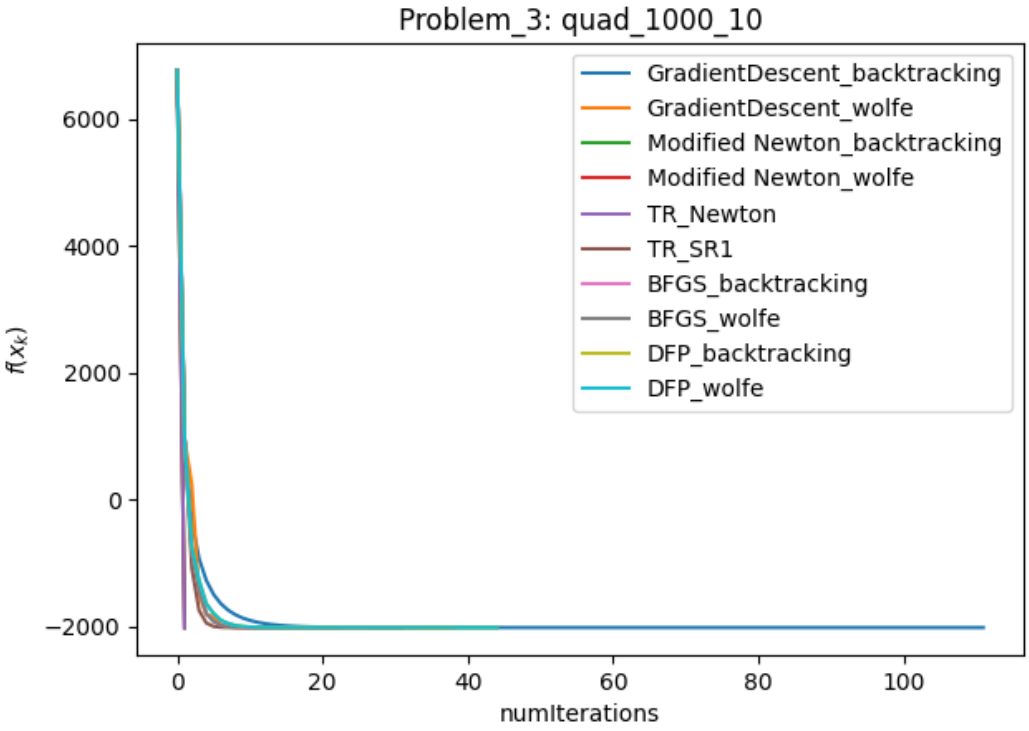
We will definitely recommend gradient descent with backtracking line search to a beginning coder as this method does not require hessian, involves fewer parameters and their tuning, and expression for descent direction is also very straightforward. Additionally, this method performs decently for convex as well as non-convex problems. However, an expert coder should work on Modified Newton with Wolfe line search as it involves the calculation of hessian, involves a large number of parameters and their tuning, and estimation of η based on implementing Cholesky decomposition. The coder should emphasize more on devising better and more time-effective mechanisms of parameter tuning then he/she can improve performance. Though we recommended Modified Newton with backtracking line search as our best choice based on heuristics, we believe Modified Newton Wolfe can also be improved with a robust parameter tuning mechanism. While working on the project, we realized that parameter tuning is a difficult task as we need to make a proper choice of parameter values in the grid. After this, when we run all combinations of the parameter grid, it becomes a large number of combinations after nesting them in for loop i.e., n^m combinations where n is the number of discrete values for one parameter and m is the total number of parameters. This can be a large number that requires significant computation time and even after that, we cannot be sure if we have not missed any better combination. We also noted that performance profiles and conventional tabular performance results/ plots may not yield the same algorithm as the best choice, rather we shall try to select the best choice algorithm by balancing the weights. In sum, we feel this course project is a great opportunity and helped us to hone our coding and optimization skills significantly while working and dealing with challenges in this project.

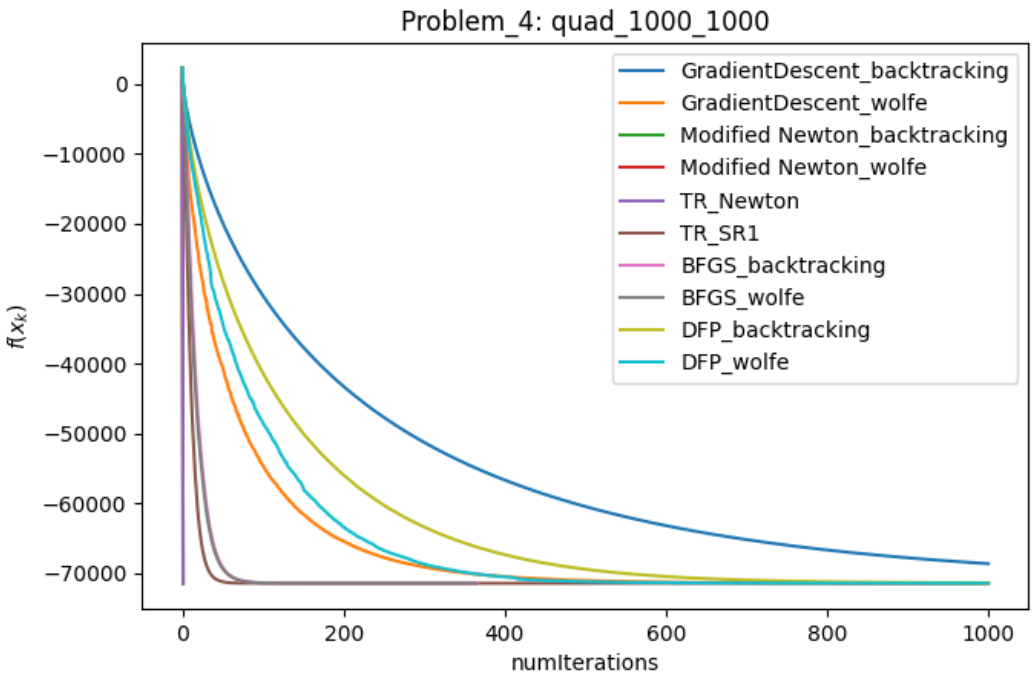
Appendices

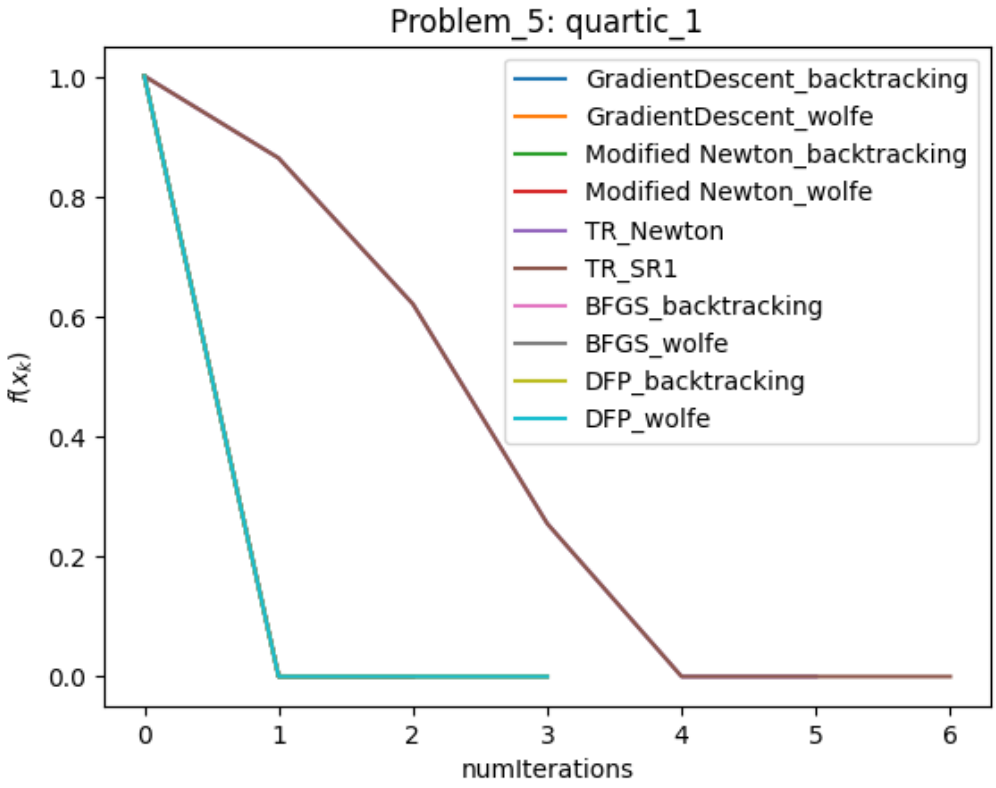
3.1 The graphs of number of iterations vs $f(x_k)$

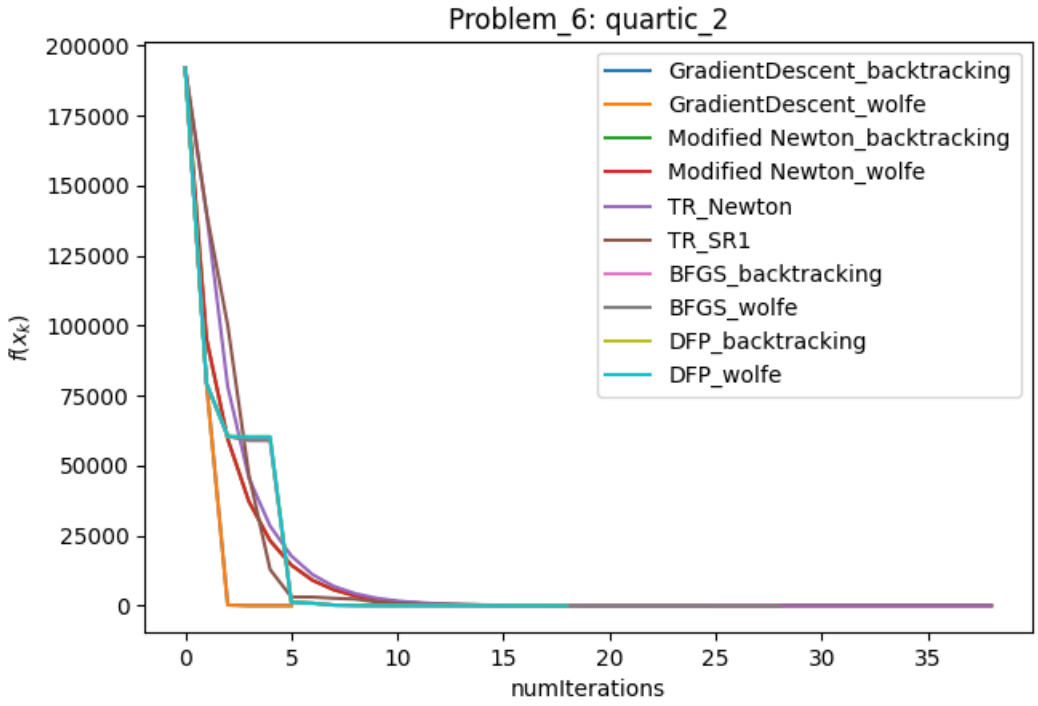


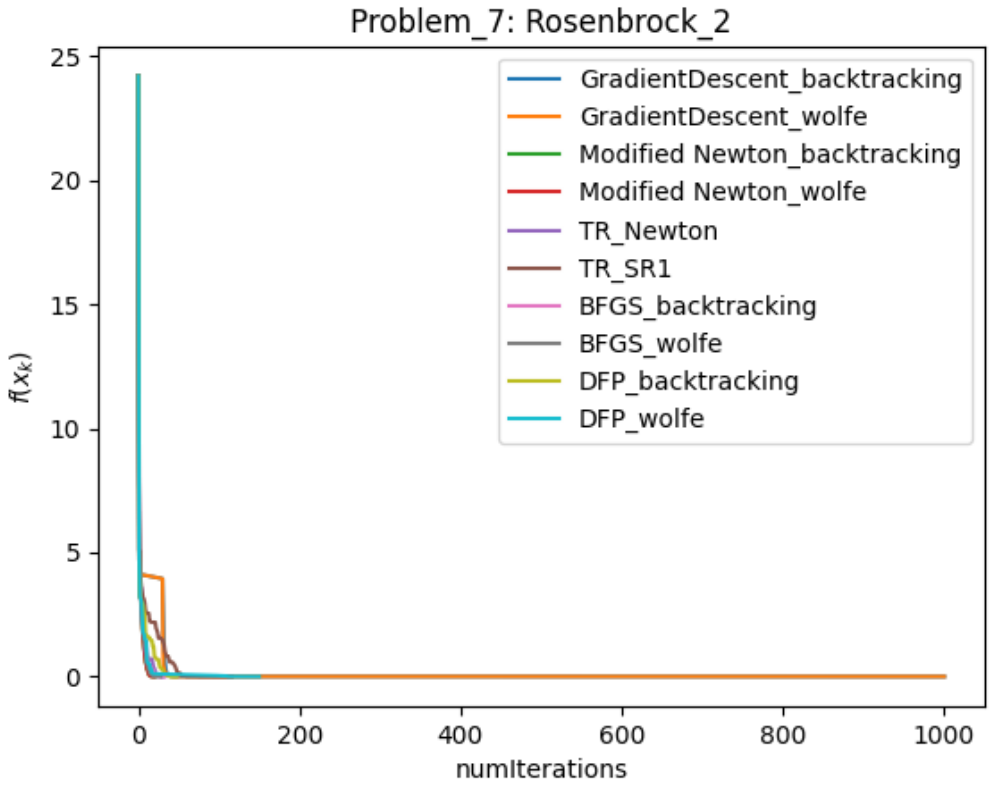

Problem_2: quad_10_1000
GradientDescent_backtracking
GradientDescent_wolfe
Modified Newton_backtracking
Modified Newton_wolfe
TR_Newton
TR_SR1
BFGS_backtracking
BFGS_wolfe
DFP_backtracking
DFP_wolfe
0
−100
−200
−300
−400
−500
−600
−700
$f(x_k)$
0
200
400
600
800
1000
numIterations

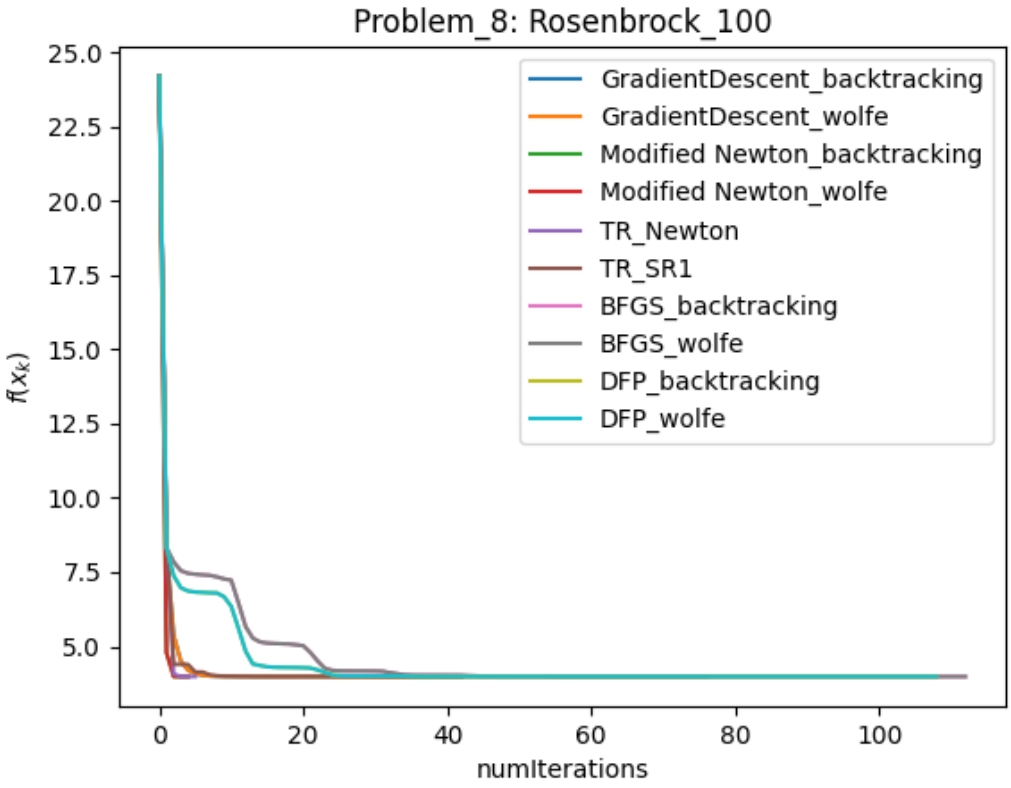

Problem_3: quad_1000_10
GradientDescent_backtracking
GradientDescent_wolfe
Modified Newton_backtracking
Modified Newton_wolfe
TR_Newton
TR_SR1
BFGS_backtracking
BFGS_wolfe
DFP_backtracking
DFP_wolfe
6000
4000
2000
0
−2000
$f(x_k)$
0
20
40
60
80
100
numIterations

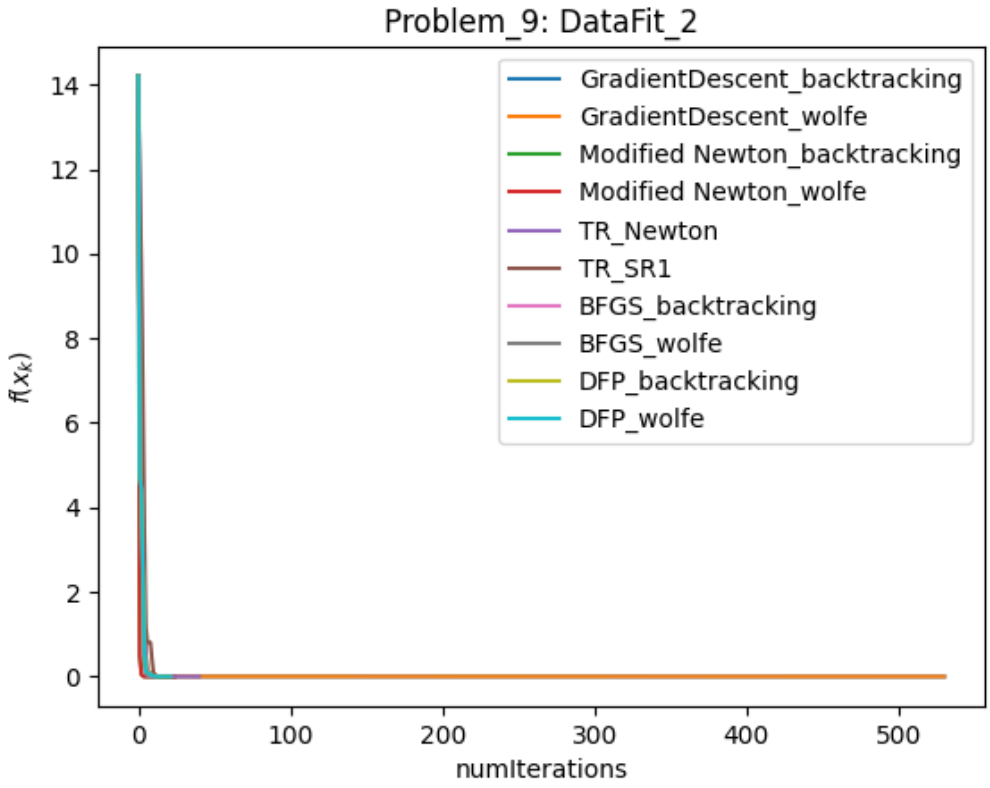

Problem_4: quad_1000_1000
GradientDescent_backtracking
GradientDescent_wolfe
Modified Newton_backtracking
Modified Newton_wolfe
TR_Newton
TR_SR1
BFGS_backtracking
BFGS_wolfe
DFP_backtracking
DFP_wolfe
0
−10000
−20000
−30000
−40000
−50000
−60000
−70000
f(x_k)
0
200
400
600
800
1000
numIterations

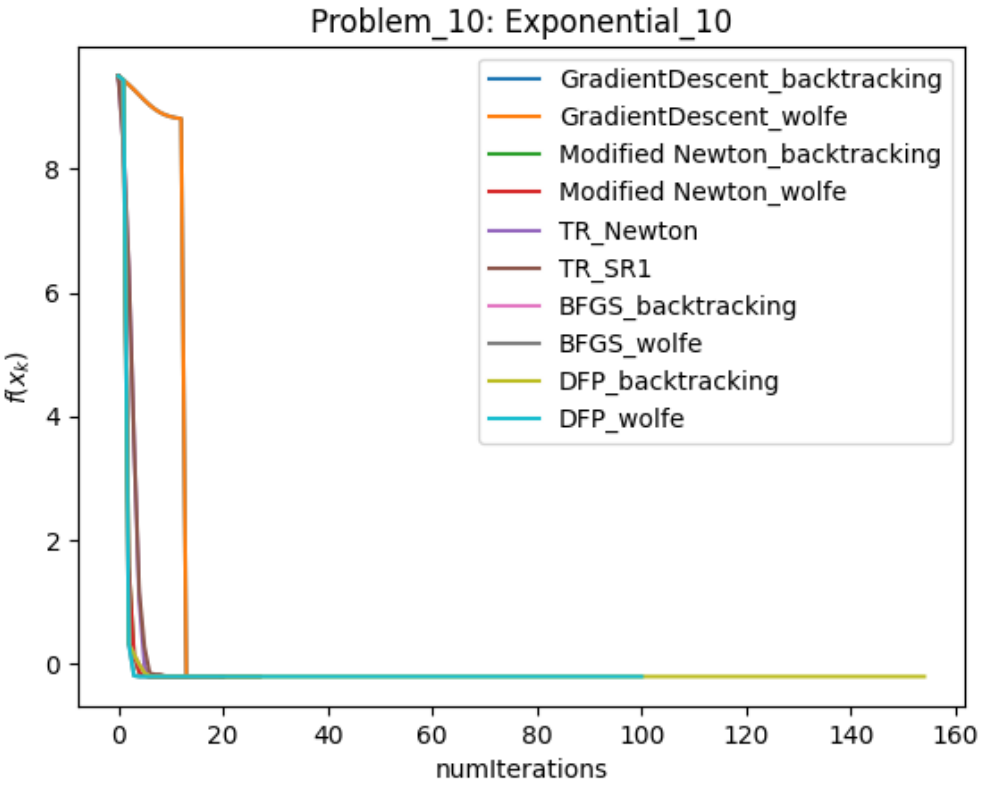

Problem_5: quartic_1
GradientDescent_backtracking
GradientDescent_wolfe
Modified Newton_backtracking
Modified Newton_wolfe
TR_Newton
TR_SR1
BFGS_backtracking
BFGS_wolfe
DFP_backtracking
DFP_wolfe
1.0
0.8
0.6
0.4
0.2
0.0
$f(x_k)$
0
1
2
3
4
5
6
numIterations

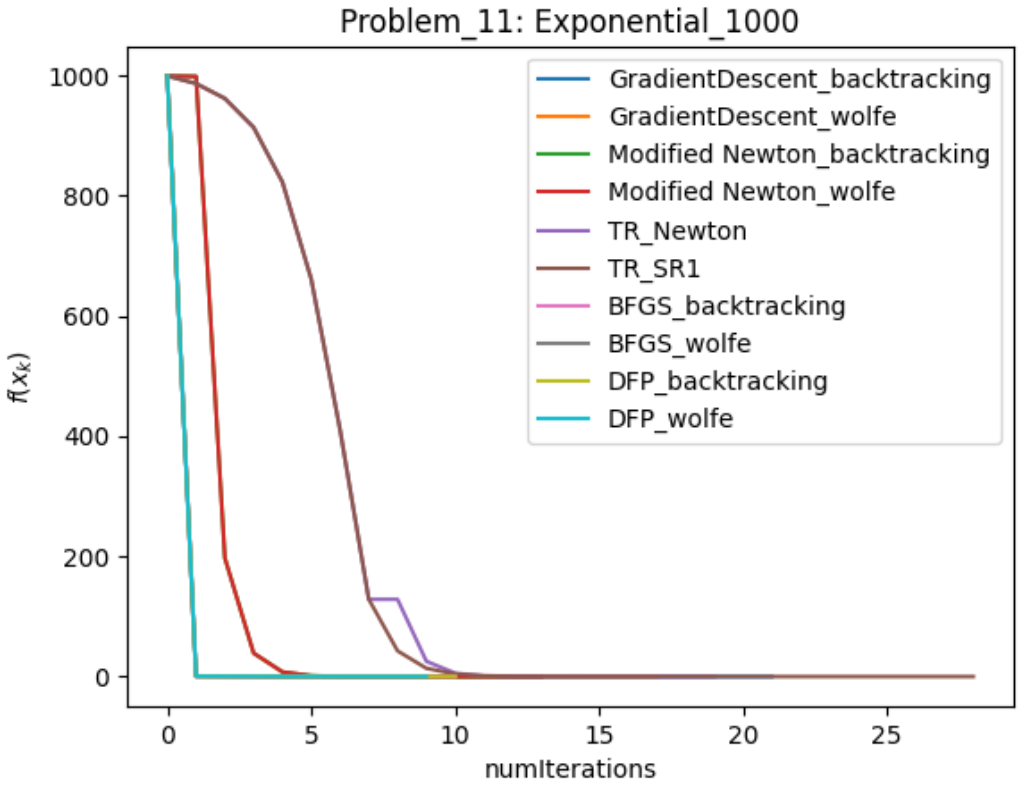

Problem_6: quartic_2
GradientDescent_backtracking
GradientDescent_wolfe
Modified Newton_backtracking
Modified Newton_wolfe
TR_Newton
TR_SR1
BFGS_backtracking
BFGS_wolfe
DFP_backtracking
DFP_wolfe
$f(x_k)$
numIterations
200000
175000
150000
125000
100000
75000
50000
25000
0
0
5
10
15
20
25
30
35

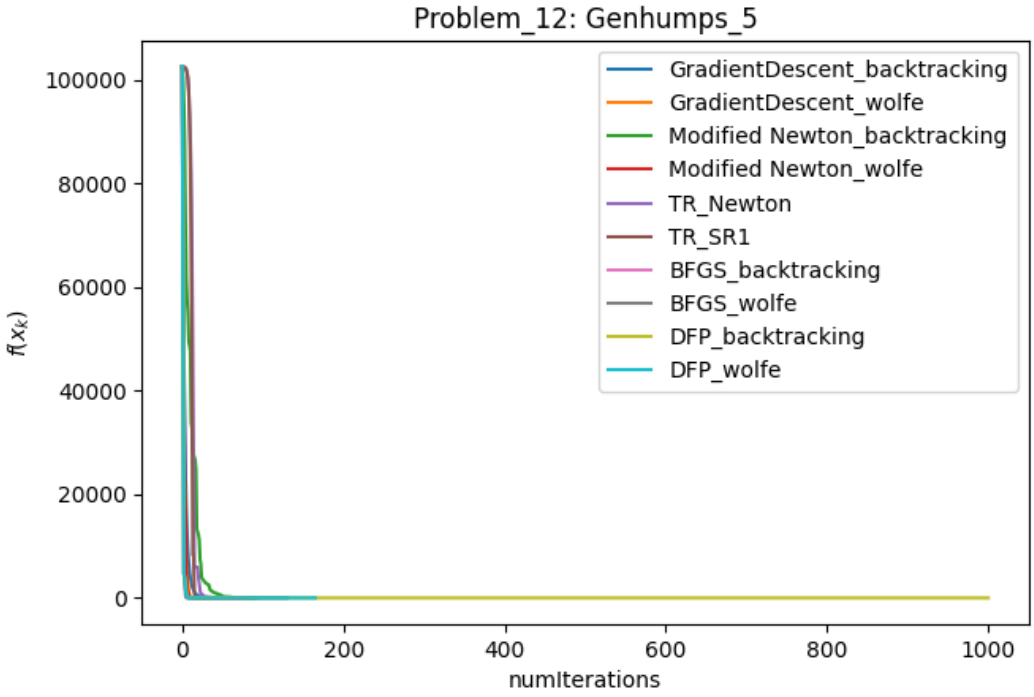

Problem_7: Rosenbrock_2
GradientDescent_backtracking
GradientDescent_wolfe
Modified Newton_backtracking
Modified Newton_wolfe
TR_Newton
TR_SR1
BFGS_backtracking
BFGS_wolfe
DFP_backtracking
DFP_wolfe
$f(x_k)$
numIterations
0
5
10
15
20
25
0
200
400
600
800
1000


Problem_8: Rosenbrock_100
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
$f(x_k)$
0
20
40
60
80
100
numIterations
GradientDescent_backtracking
GradientDescent_wolfe
Modified Newton_backtracking
Modified Newton_wolfe
TR_Newton
TR_SR1
BFGS_backtracking
BFGS_wolfe
DFP_backtracking
DFP_wolfe

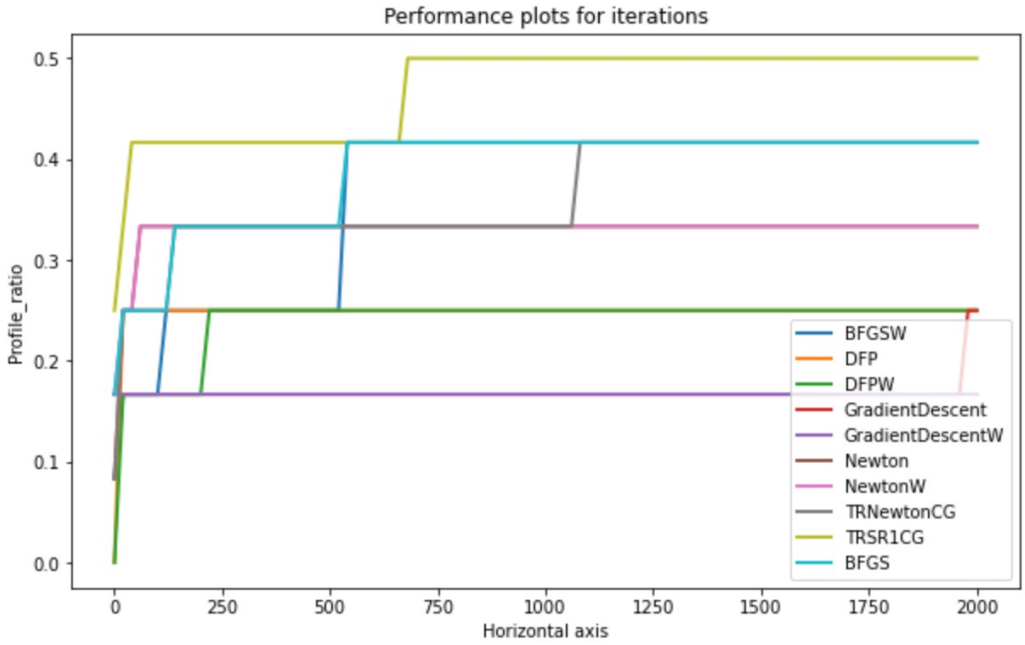

Problem_9: DataFit_2
GradientDescent_backtracking
GradientDescent_wolfe
Modified Newton_backtracking
Modified Newton_wolfe
TR_Newton
TR_SR1
BFGS_backtracking
BFGS_wolfe
DFP_backtracking
DFP_wolfe
$f(x_k)$
numIterations
0
2
4
6
8
10
12
14
0
100
200
300
400
500


Problem_10: Exponential_10
GradientDescent_backtracking
GradientDescent_wolfe
Modified Newton_backtracking
Modified Newton_wolfe
TR_Newton
TR_SR1
BFGS_backtracking
BFGS_wolfe
DFP_backtracking
DFP_wolfe
f(x_k)
0
2
4
6
8
0
20
40
60
80
100
120
140
160
numIterations


Problem_11: Exponential_1000
GradientDescent_backtracking
GradientDescent_wolfe
Modified Newton_backtracking
Modified Newton_wolfe
TR_Newton
TR_SR1
BFGS_backtracking
BFGS_wolfe
DFP_backtracking
DFP_wolfe
1000
800
600
400
200
0
$f(x_k)$
0
5
10
15
20
25
numIterations


Problem_12: Genhumps_5
GradientDescent_backtracking
GradientDescent_wolfe
Modified Newton_backtracking
Modified Newton_wolfe
TR_Newton
TR_SR1
BFGS_backtracking
BFGS_wolfe
DFP_backtracking
DFP_wolfe
100000
80000
60000
40000
20000
0
$f(x_k)$
0
200
400
600
800
1000
numIterations

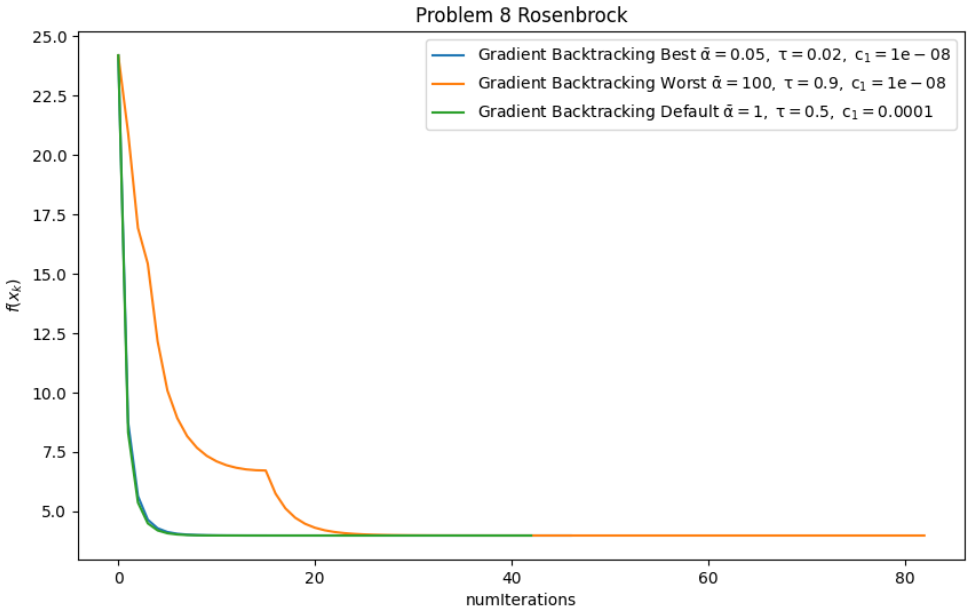
3.2 Performance Profile

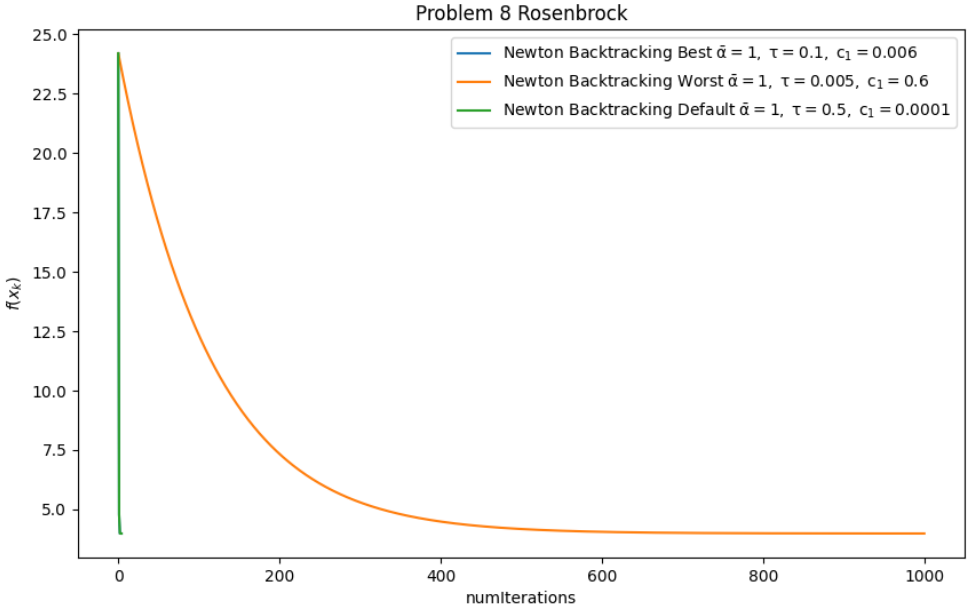


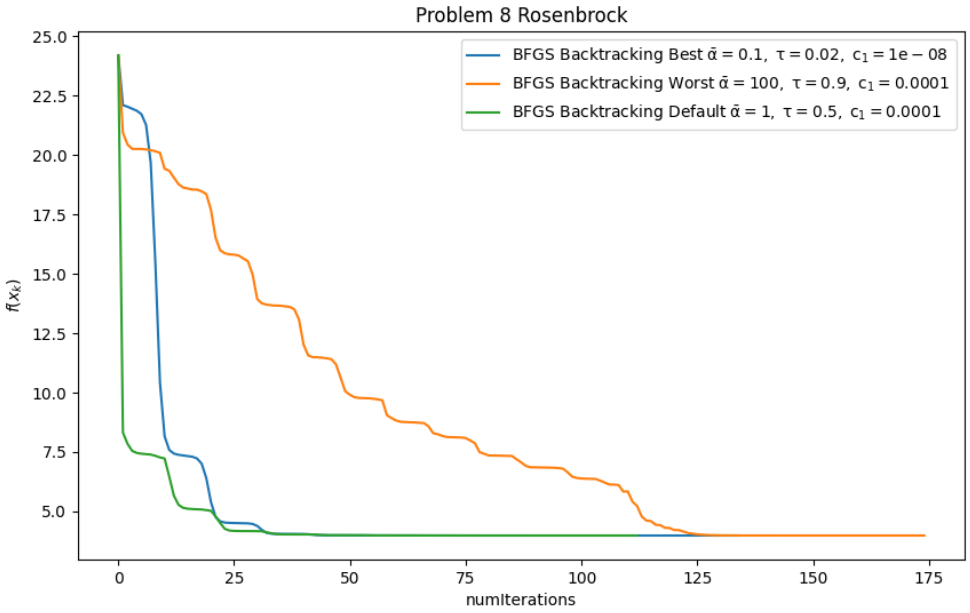

```
Best measure ranking:
              TRSR1CG solved 25.00 percent of the problems with the best performance measure
                 BFGS solved 16.67 percent of the problems with the best performance measure
              NewtonW solved 16.67 percent of the problems with the best performance measure
    GradientDescentW solved 16.67 percent of the problems with the best performance measure
     GradientDescent solved 16.67 percent of the problems with the best performance measure
          TRNewtonCG solved  8.33 percent of the problems with the best performance measure
              Newton solved  8.33 percent of the problems with the best performance measure
               BFGSW solved  8.33 percent of the problems with the best performance measure
                DFPW solved  0.00 percent of the problems with the best performance measure
                 DFP solved  0.00 percent of the problems with the best performance measure

Reliability ranking:
             TRSR1CG solved 50.00 percent of the problems with ratio less than or equal to tau_max = 2000
                BFGS solved 41.67 percent of the problems with ratio less than or equal to tau_max = 2000
          TRNewtonCG solved 41.67 percent of the problems with ratio less than or equal to tau_max = 2000
               BFGSW solved 41.67 percent of the problems with ratio less than or equal to tau_max = 2000
            NewtonW solved 33.33 percent of the problems with ratio less than or equal to tau_max = 2000
             Newton solved 33.33 percent of the problems with ratio less than or equal to tau_max = 2000
      GradientDescent solved 25.00 percent of the problems with ratio less than or equal to tau_max = 2000
                DFPW solved 25.00 percent of the problems with ratio less than or equal to tau_max = 2000
                 DFP solved 25.00 percent of the problems with ratio less than or equal to tau_max = 2000
    GradientDescentW solved 16.67 percent of the problems with ratio less than or equal to tau_max = 2000
```

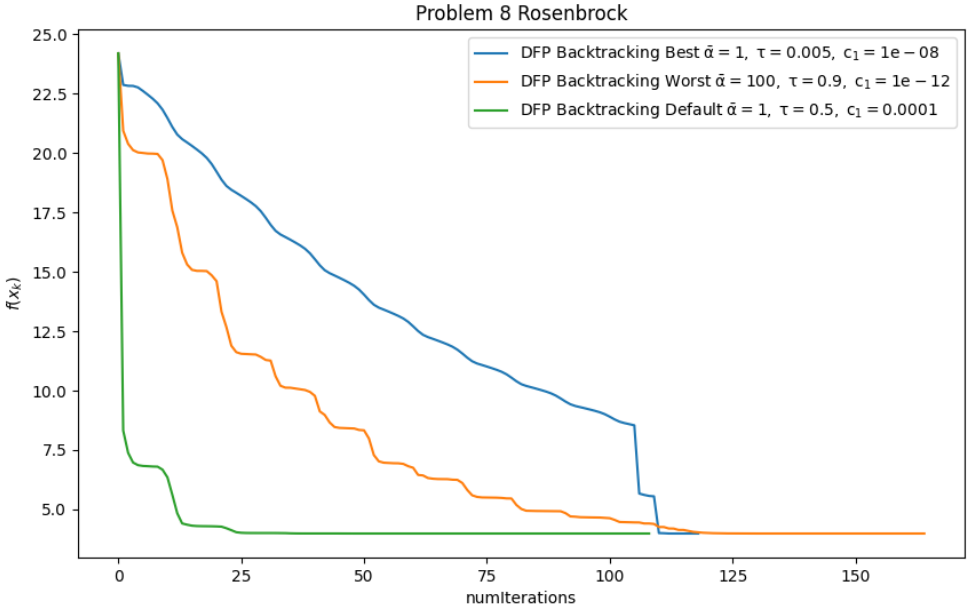
3.3 Graphs of Parameter Tuning

Backtracking

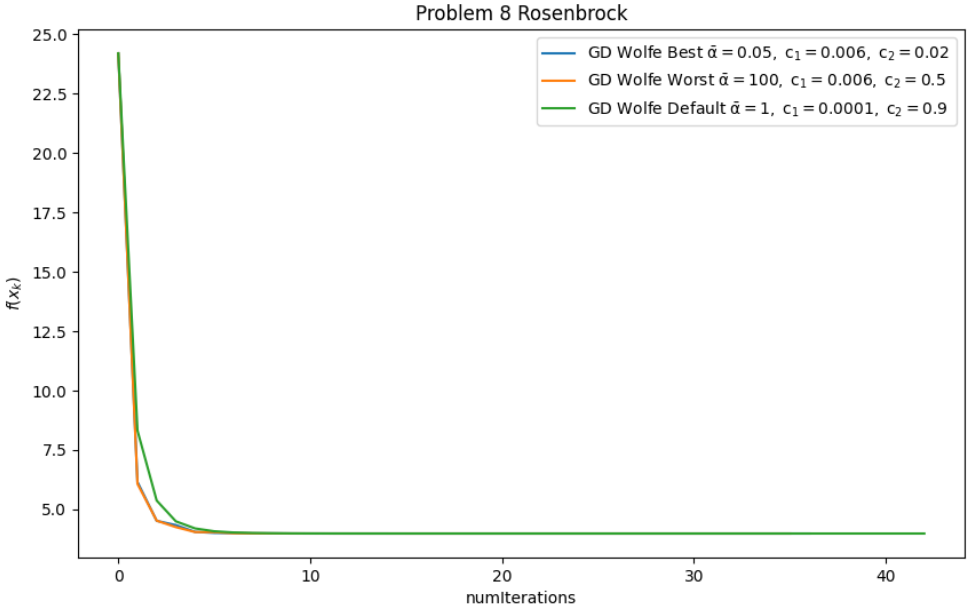

Problem 8 Rosenbrock
Gradient Backtracking Best $\bar{\alpha} = 0.05,\ \tau = 0.02,\ c_1 = 1e-08$
Gradient Backtracking Worst $\bar{\alpha} = 100,\ \tau = 0.9,\ c_1 = 1e-08$
Gradient Backtracking Default $\bar{\alpha} = 1,\ \tau = 0.5,\ c_1 = 0.0001$
$f(x_k)$
numIterations

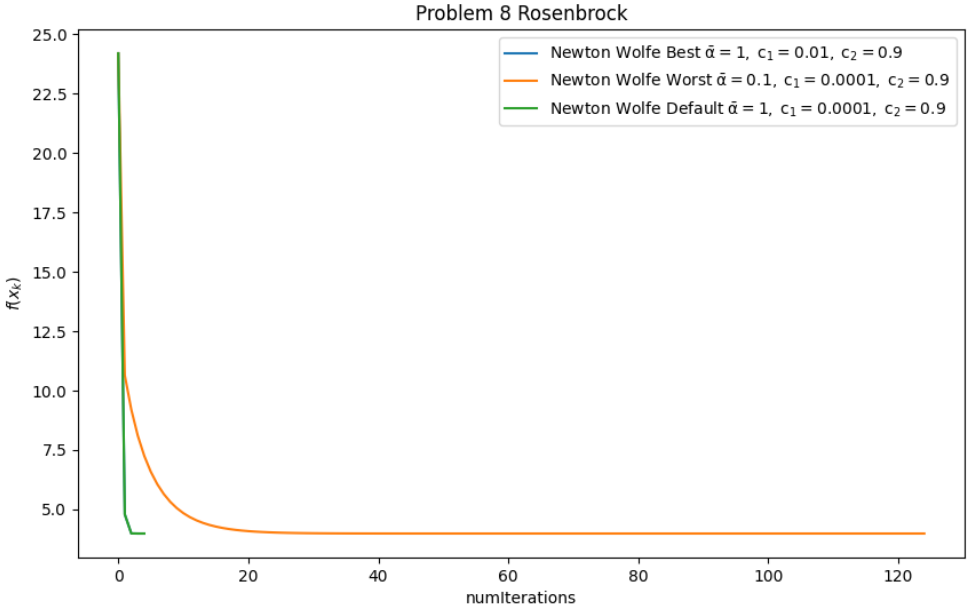

Problem 8 Rosenbrock
Newton Backtracking Best $\bar{\alpha} = 1,\ \tau = 0.1,\ c_1 = 0.006$
Newton Backtracking Worst $\bar{\alpha} = 1,\ \tau = 0.005,\ c_1 = 0.6$
Newton Backtracking Default $\bar{\alpha} = 1,\ \tau = 0.5,\ c_1 = 0.0001$
$f(x_k)$
numIterations

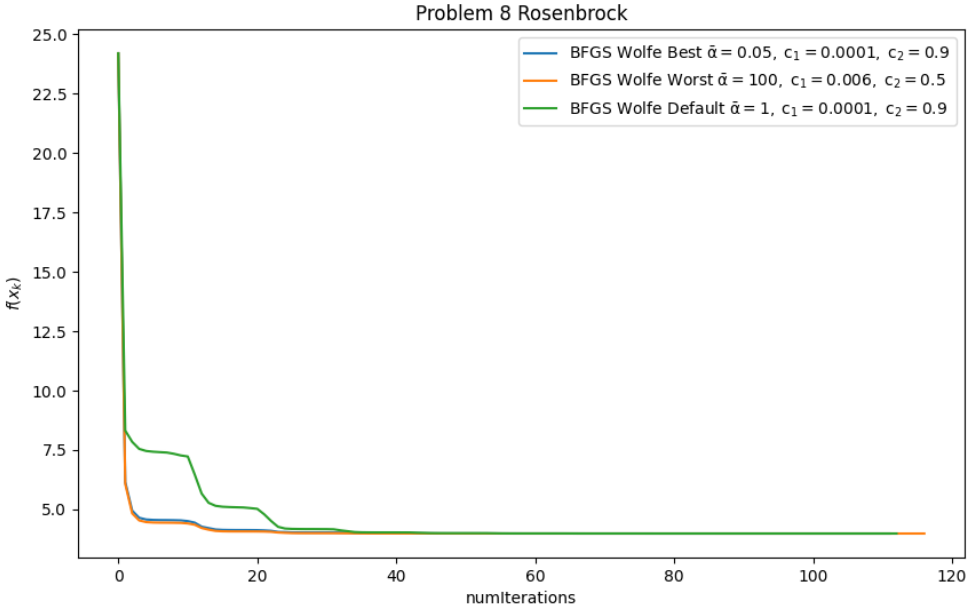

Problem 8 Rosenbrock
BFGS Backtracking Best $\bar{\alpha} = 0.1,\ \tau = 0.02,\ c_1 = 1e-08$
BFGS Backtracking Worst $\bar{\alpha} = 100,\ \tau = 0.9,\ c_1 = 0.0001$
BFGS Backtracking Default $\bar{\alpha} = 1,\ \tau = 0.5,\ c_1 = 0.0001$
$f(x_k)$
numIterations

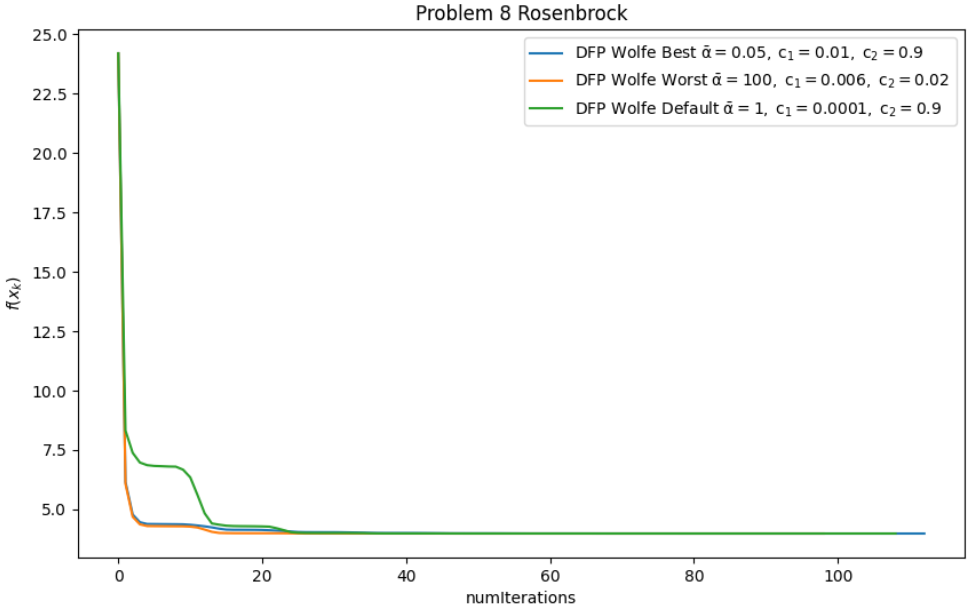

Problem 8 Rosenbrock
DFP Backtracking Best $\bar{\alpha} = 1,\ \tau = 0.005,\ c_1 = 1e-08$
DFP Backtracking Worst $\bar{\alpha} = 100,\ \tau = 0.9,\ c_1 = 1e-12$
DFP Backtracking Default $\bar{\alpha} = 1,\ \tau = 0.5,\ c_1 = 0.0001$
$f(x_k)$
numIterations

Wolfe


Problem 8 Rosenbrock
GD Wolfe Best $\bar{\alpha} = 0.05,\ c_1 = 0.006,\ c_2 = 0.02$
GD Wolfe Worst $\bar{\alpha} = 100,\ c_1 = 0.006,\ c_2 = 0.5$
GD Wolfe Default $\bar{\alpha} = 1,\ c_1 = 0.0001,\ c_2 = 0.9$
$f(x_k)$
numIterations


Problem 8 Rosenbrock
Newton Wolfe Best $\bar{\alpha} = 1,\ c_1 = 0.01,\ c_2 = 0.9$
Newton Wolfe Worst $\bar{\alpha} = 0.1,\ c_1 = 0.0001,\ c_2 = 0.9$
Newton Wolfe Default $\bar{\alpha} = 1,\ c_1 = 0.0001,\ c_2 = 0.9$
$f(x_k)$
numIterations


Problem 8 Rosenbrock
BFGS Wolfe Best $\tilde{\alpha} = 0.05,\ c_1 = 0.0001,\ c_2 = 0.9$
BFGS Wolfe Worst $\tilde{\alpha} = 100,\ c_1 = 0.006,\ c_2 = 0.5$
BFGS Wolfe Default $\tilde{\alpha} = 1,\ c_1 = 0.0001,\ c_2 = 0.9$
$f(x_k)$
numIterations


Problem 8 Rosenbrock
DFP Wolfe Best $\tilde{\alpha} = 0.05,\ c_1 = 0.01,\ c_2 = 0.9$
DFP Wolfe Worst $\tilde{\alpha} = 100,\ c_1 = 0.006,\ c_2 = 0.02$
DFP Wolfe Default $\tilde{\alpha} = 1,\ c_1 = 0.0001,\ c_2 = 0.9$
$f(x_k)$
numIterations