Katherine Ahn (kathahn@umich.edu), Seonho Woo (clairewo@umich.edu)

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

---

1 Introduction

This project investigates the algorithms' performances for finding the minimizer of unconstrained continuous problems. There are two main methods for determining the search direction of such algorithms: line search methods and trust region methods. We implemented several algorithms from both groups of algorithms and conducted an in-depth analysis of the performances of each algorithm on the given set of problems. Additionally, we further investigated the BFGS algorithm regarding whether and how the choice of the initial Hessian approximation has an impact on the algorithmic performance.

2 Algorithm Descriptions

2.1 Line Search Methods

The line search methods first find the direction of the functional descent via gradient and Hessian of the objective function. It later decides the step length afterward. There exist several principled approaches to finding the appropriate step length given a search direction. In this project, we implemented **Backtracking** and **Wolfe Line Search**.

Backtracking

Given the search direction d_k at x_k, the backtracking line search algorithm finds α_k that satisfies the following where $c_1 > 0$ is a user-defined parameter:

$$f(x_k + \alpha_{\mathbf{k}} d_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T d_k$$

This condition ensures sufficient reduction of the functional value at the new iterate.

Wolfe Line Search

The Wolfe line search finds α_k such that satisfies the following two conditions given the search direction d_k at x_k, where $0 < c_1 < c_2 < 1$.

$$f(x_k + \alpha_{\mathbf{k}} d_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T d_k$$
$$\nabla f(x_k + \alpha_{\mathbf{k}} d_k)^T d_k \geq c_2 \nabla f(x_k)^T d_k$$

It allows the algorithm to move while ensuring sufficient functional value reduction at the new iterate.

2.1.1 Gradient Descent

This algorithm takes steps in the negative gradient direction of the function to find the minimum. The step size is controlled by a learning rate parameter, and it iterates until it meets the termination conditions, such as maximum iterations or termination tolerance. If the function is strongly convex, the minimum

you find is always the global minimum, but if not, it would converge to the local minimum. This can also be controlled by initializing hyperparameters and learning rates to avoid getting local optimum. The method has a theoretical guarantee of converging to the stationary point.

2.1.2 Modified Newton's Method

Modified Newton's method uses second-order information (hessian) to update the search direction in each iteration. This is the biggest improvement from the Gradient Descent method, which is only using first-order information (gradient). Before it uses Hessian to update, it checks if it is positive definite. If not (nonconvex), then use a Hessian approximation for a positive definite. Specifically, the algorithm undergoes a Cholesky-decomposition subroutine to find the Hessian approximation(Modified Newton's method). The method theoretically guarantees a local quadratic convergence rate under some assumptions.

2.1.3 BFGS & DFP

BFGS and DFP are quasi-Newton methods. The quasi-Newton methods update and use the approximation of the Hessian matrix to compute the search direction at every iteration. The update of the Hessian matrix involves solving the secant equation, defined by current and previous gradients, and iterates, resulting in less computational cost than Newton's method. Moreover, each update goes through a rank-2 update from the current Hessian approximation for both algorithms while retaining the symmetricity. BFGS and DFP achieve a superlinear convergence rate when the objective function is strongly convex and has a Lipschitz continuous gradient.

2.2 Trust Region Methods

The trust region method first constructs a quadratic model, trust region, such that it approximates the objective function at each iterate within a radius. At each iteration, the algorithm solves a subproblem: finding the search direction that minimizes the functional value of the model given a constraint on the length of the direction. Moreover, the algorithm adjusts the trust region radius based on the performance of the model in reducing the functional value. And it tackles nonconvexity by using a global convergence test that ensures the existence of a minimizer of the objective function within a certain radius of the current point. The computation between steps involves computing the gradient and Hessian of the objective function, constructing a quadratic approximation of the objective function, and solving its subproblem.

2.2.1 TRNewtonCG & TRSR1CG

Trust Region Newton with Conjugate Gradient (CG) subproblem solver uses second-order information to construct a quadratic approximation of the objective function, which is used to solve a Trust Region subproblem in each iteration. The CG method efficiently solves the Trust Region subproblem by computing conjugate directions and step sizes that minimize the quadratic model subject to the Trust Region constraint. TRNewton computes the exact Hessian matrix to update at each iteration, but TRSR1 computes the gradient of the objective function and updates the quasi-Newton approximation for the Hessian matrix using rank-one updates.

3 Algorithm Implementation

options	values
`term_tol`	1e-6
`max_iterations`	1e3
`c_1_ls`	0.1
`c_2_ls`	0.2
`c_1_tr`	0.3
`c_2_tr`	0.7
`term_tol_CG`	1e-6
`max_iterations_CG`	1e3

Table 1: Table of default options

Written values in the default options table (Table 1) were determined by the performance of overall algorithms and problems. We conducted a hyperparameter search and some tunings within these ranges: `c_1_ls` of [0.5, 0.2, 0.1, 1e-2, 1e-3], `c_2_ls` of [0.7, 0.5, 0.3, 0.2, 0.1], `c_1_tr` of [0.1, 0.3], `c_2_tr` of [0.5, 0.7], `term_tol_CG` of [1e-3, 1e-6], and `max_iterations_CG` of 100 and 1000. As a result, we fixed the values in Table 1 by finding out some ranges where most problems and algorithms converge or have converging behaviors. Moreover, we set the initial Hessian approximation for the Quasi-Newton methods to the identity matrix.

3.1 Results and Observations

Here we provide results of implementing ten algorithms on twelve given problems. The parameters are set to the default parameter values. The default parameter values are chosen based on tuning as described above.

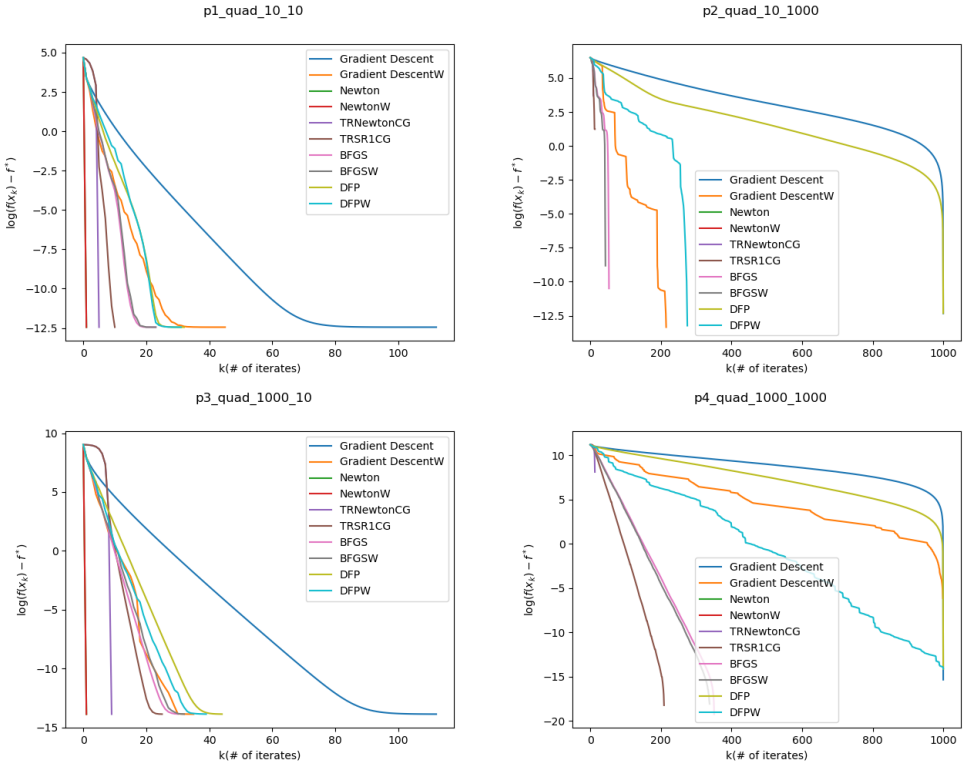


Fig 1. Quadratic Problems

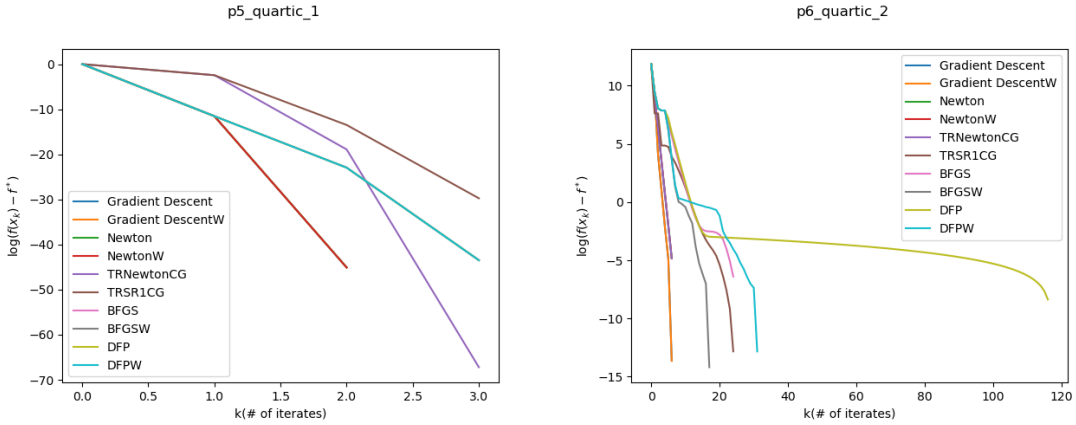


Fig 2. Quartic Problems

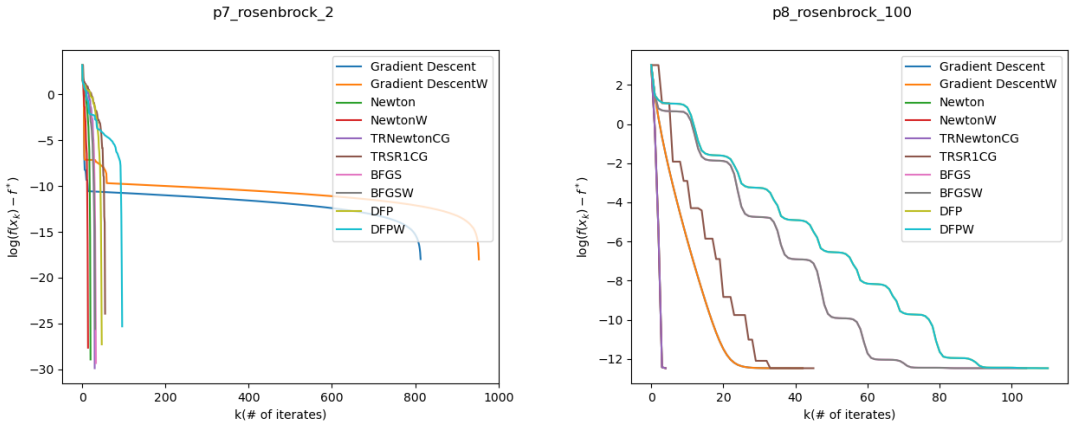


Fig 3. Rosenbrock Problems

As shown in the convergence plots above, for many of the problems, the convergence rate for all the algorithms matches the theoretical results. Modified Newton's method shows a quadratic convergence rate, the quasi-Newton method with Wolfe line search shows a superlinear convergence rate, and Trust region method algorithms converge quadratically/superlinearly. Specifically, in solving quadratic problems with various settings of dimension and condition numbers, GD algorithms converge slower with larger condition numbers, and the number of iterations increases as the dimension of the problem increases, as expected. They also converge linearly if they are well-conditioned, as most of the other algorithms behave.

For well-conditioned strongly convex quadratic functions(problems 1 and 3), the gradient descent with backtracking shows a linear convergence rate. According to the theory, the gradient descent with fixed step size has a linear convergence rate for strongly convex functions.

As mentioned earlier, DFP with Wolfe line search generally shows a superlinear convergence rate. On the other hand, DFP with Armijo backtracking line search demonstrates a sublinear convergence rate for a few problems. In fact, DFP with backtracking showing a sublinear convergence rate does not converge within the pre-specified max iterations. Here we conclude Wolfe line search is essential for DFP to converge. We also observe Wolfe line search converges to the solution with fewer iterations than the Armijo backtracking line search, and this is a general trend in a difference between the two line search methods we use; the Wolfe line search also has a larger step size than the Armijo backtracking line search. Convergence plots on implementations of problems 9 to 12 (DataFit, Exponential, Genhumps) can be found in **Appendix A**.

Problem 1	Problem 2	Problem 3	Problem 4	Problem 5	Problem 6
quad_10_10	quad_10_1000	quad_1000_10	quad_1000_1000	quartic_1	quartic_2
Problem 7	Problem 8	Problem 9	Problem 10	Problem 11	Problem 12
Rosenbrock_2	Rosenbrock_100	Datafit_2	Exponential_10	Exponential_1000	Genhumps_5

Table 2: Problem numbers and features

	Convergence Algorithms									
Prob#	GD	GDW	New	NewW	TRNCG	TRSR1CG	BFGS	BFGSW	DFP	DFPW
1	-2.47088e+01	-2.47088e+01	-2.47088e+01	-2.47088e+01	-2.47088e+01	-2.47088e+01	-2.47088e+01	-2.47088e+01	-2.47088e+01	-2.47088e+01
2	-6.66554e+02	-6.73629e+02	-6.73629e+02	-6.73629e+02	-6.73629e+02	-6.73629e+02	-6.73629e+02	-6.73629e+02	-6.73331e+02	-6.73629e+02
3	-2.01497e+03	-2.01497e+03	-2.01497e+03	-2.01497e+03	-2.01497e+03	-2.01497e+03	-2.01497e+03	-2.01497e+03	-2.01497e+03	-2.01497e+03
4	-6.86658e+04	-7.14628e+04	-7.14635e+04	-7.14635e+04	-7.14635e+04	-7.14635e+04	-7.14635e+04	-7.14635e+04	-7.14073e+04	-7.14635e+04
5	2.67096e-20	2.67096e-20	2.55329e-20	2.55329e-20	6.45833e-30	1.19933e-13	1.31539e-19	1.31539e-19	1.30037e-19	1.30037e-19
6	7.51195e-04	7.51195e-04	5.66030e-04	5.66030e-04	5.66030e-04	1.51268e-03	9.56861e-04	1.53069e-03	2.44790e-03	1.46266e-03
7	7.42385e-06	1.85706e-05	2.75421e-13	9.94521e-13	1.06114e-13	4.05923e-11	1.86604e-13	7.67163e-12	1.42922e-12	1.01753e-11
8	3.98662e+00	3.98662e+00	3.98662e+00	3.98662e+00	3.98662e+00	3.98662e+00	3.98662e+00	3.98662e+00	3.98662e+00	3.98662e+00
9	1.00880e-09	1.11888e-09	6.65655e-11	4.55091e-11	5.61228e-11	7.27707e-12	3.80213e-12	9.31303e-15	3.71532e-10	7.82985e-15
10	-2.05573e-01	-2.05573e-01	-2.05573e-01	-2.05573e-01	-2.05573e-01	-2.05573e-01	-2.05573e-01	-2.05573e-01	-2.05573e-01	-2.05573e-01
11	-2.05573e-01	-2.05573e-01	-2.05569e-01	-2.05573e-01	-2.05566e-01	-2.05573e-01	-2.05573e-01	-2.05573e-01	-2.05573e-01	-2.05573e-01
12	8.71841e-08	3.67920e-08	2.09315e-11	2.64545e-14	2.66530e-08	4.30377e-08	1.62901e-08	8.94316e-09	1.37736e+00	4.45153e-08

Table 3: f^* value at termination (optimum if converges)

As we can see from Table 3, in terms of minimum f^* value, in problems 1 to 4, which are quadratic problems with different dimensions and condition numbers of 10 and 1000, Modified Newton's methods using either backtracking or Wolfe line search demonstrate a good performance in getting the minimum optimal function value. It constantly shows the best (smallest) f^* value compared to other algorithms with one iteration. We can infer that Newton's method has optimal performance in quadratic problems based on its computation algorithms. Comparatively, in problems 5 to 8 (quartic and rosenbrock), we find that Trust Region Newton's method with CG subproblem solver performs the best in minimizing f optimal value, empirically. In the Datafit problem (problem 9), DFP performs best, and in problems 10 to 12, there are no dramatic differences in the f^* value between algorithms, though Newton's method has slightly smaller f^* values than others.

	Convergence Algorithms									
Prob#	GD	GDW	New	NewW	TRNCG	TRSR1CG	BFGS	BFGSW	DFP	DFPW
1	7.058e-06	7.303e-06	1.499e-15	1.499e-15	8.571e-14	5.015e-06	3.008e-06	5.080e-06	3.965e-06	7.075e-06
2	9.402e-02	1.542e-06	1.532e-14	1.532e-14	8.880e-08	7.773e-10	3.614e-07	9.093e-08	3.117e-01	7.125e-07
3	9.193e-06	8.601e-06	1.521e-14	1.521e-14	9.061e-08	4.995e-06	9.457e-06	9.470e-06	9.506e-06	9.675e-06
4	7.319e-01	5.702e-02	3.681e-13	3.681e-13	1.617e-07	8.518e-06	8.871e-06	7.395e-06	5.514e-01	1.961e-05
5	1.570e-10	1.570e-10	1.535e-10	1.535e-10	2.653e-15	3.566e-07	4.217e-10	4.217e-10	4.193e-10	4.193e-10
6	1.755e-01	1.755e-01	1.566e-01	1.566e-01	1.566e-01	3.165e-01	2.421e-01	3.174e-01	3.206e-01	2.443e-01
7	3.832e-03	5.671e-03	8.253e-07	1.842e-05	2.837e-07	4.017e-05	9.717e-06	7.077e-05	2.567e-05	1.146e-04
8	2.058e-04	2.058e-04	6.453e-07	6.453e-07	9.114e-07	3.083e-05	1.849e-04	1.849e-04	5.757e-06	5.757e-06
9	2.270e-05	2.389e-05	2.496e-05	2.063e-05	2.292e-05	7.849e-06	1.841e-05	1.016e-07	1.745e-05	6.618e-07
10	2.166e-06	3.440e-06	1.817e-06	2.998e-06	1.784e-06	3.668e-06	8.964e-07	8.966e-07	8.986e-07	8.988e-07
11	2.166e-06	3.440e-06	1.803e-06	8.146e-07	3.020e-06	3.346e-06	4.657e-09	8.107e-07	4.660e-09	8.111e-07
12	9.337e-05	8.477e-05	2.590e-06	6.898e-08	7.301e-05	8.236e-05	5.730e-05	3.215e-05	6.316e-01	8.261e-05

Table 4: $\|\nabla f\|$ at termination (optimum if converges)

Generally, a norm of g, ($\|\nabla f\|$), decreases and approaches zero in the end as it reaches its convergence. We can see that most algorithms in all problems converge to optimal and a few cases with nonzero g–norm are also decreasing (not monotonically but in overall trend). Thus, we can assume that it will converge within larger maximum iteration numbers (instead of using 1000 as in this implementation).

	Convergence Algorithms									
Prob#	GD	GDW	New	NewW	TRNCG	TRSR1CG	BFGS	BFGSW	DFP	DFPW
1	112	45	1	1	5	10	23	23	32	31
2	1000	330	1	1	10	13	53	41	1000	278
3	112	35	1	1	9	25	30	32	44	39
4	1000	1000	1	1	14	209	362	339	1000	1000
5	2	2	2	2	3	3	3	3	3	3
6	6	6	7	7	7	24	25	17	117	31
7	1000	1000	20	14	30	55	33	31	47	96
8	42	42	4	4	4	45	104	104	110	110
9	435	470	37	9	38	18	14	14	21	16
10	21	23	13	8	12	17	10	6	10	6
11	21	23	13	13	15	24	10	6	10	6
12	178	20	124	33	103	49	39	31	1000	34

Table 5: Number of iterations

In most cases, Modified Newton's methods have the fewest iterations by their convergence. As expected, Gradient Descent Algorithms with any line search take more iterations to converge or even it should exceed the maximum iteration of 1000 to reach its convergence. Within a set of the same convergence algorithms using different line search methods: backtracking and Wolfe, Wolfe line search is observed to have a more significant step size, leading the algorithm to converge in fewer iterations. We can observe some extreme cases in problems 4 and 7 that do not converge until they reach their maximum iterations of 1000. Still, the Gradient Descent algorithms are expected to converge by the theory.

Moreover, DFP performs well in some problems but takes a long time and iterations to converge (prone to divergence) in a few other problems (i.e., 2, 4, and 12). We can guess that DFP algorithms may be more sensitive to the problem settings (i.e., dimension and condition number) than other algorithms from our empirical results.

	Convergence Algorithms									
Prob#	GD	GDW	New	NewW	TRNCG	TRSR1CG	BFGS	BFGSW	DFP	DFPW
1	0.050	0.094	0.001	0.001	0.006	0.009	0.012	0.038	0.016	0.061
2	0.457	0.591	0.001	0.001	0.008	0.010	0.026	0.072	0.478	0.499
3	9.920	13.177	0.256	0.291	1.161	3.330	4.890	14.286	7.515	19.789
4	94.582	431.145	0.300	0.326	2.147	39.949	69.538	199.129	201.407	542.086
5	0.001	0.000	0.000	0.000	0.001	0.001	0.000	0.000	0.000	0.000
6	0.002	0.002	0.001	0.001	0.002	0.005	0.003	0.004	0.012	0.008
7	0.082	0.090	0.001	0.002	0.003	0.006	0.002	0.003	0.003	0.010
8	0.059	0.070	0.005	0.006	0.020	0.066	0.225	0.227	0.178	0.192
9	0.048	0.055	0.005	0.002	0.006	0.003	0.002	0.003	0.002	0.003
10	0.002	0.006	0.002	0.002	0.002	0.003	0.001	0.001	0.001	0.001
11	0.003	0.010	0.813	0.791	0.035	0.109	0.642	0.496	0.660	0.482
12	0.020	0.006	0.037	0.015	0.021	0.012	0.005	0.008	0.106	0.008

Table 6: Computational Cost (time in seconds)

In short, computational cost (times taken to compute per iteration) is mainly in a similar range of small values. However, there are a few outliers with extremely high computational costs: problems 3 and 4. Compared to other normal-ranged values of costs, they are distinguishable with their high dimensions, and it is probably the factor that affects the computational cost of algorithms the most, as expected.

3.2 Discussion

3.2.1 Algorithm of Choice

Under the assumption of a computable Hessian matrix and within a reasonable range of dimension/condition numbers as in this experiment, *Modified Newton's method with Wolfe Line Search* would be our algorithm of choice. In terms of computational cost, Modified Newton's method uses the computed Hessian matrix if it is positive definite; otherwise, it uses approximation by adding some folds of identity matrix on its Hessian so that it can be positive definite. These sets of action take a high complexity and make the algorithm has a high computational cost in each iteration. However, as it goes through some reasonable computational actions, it usually finds the best descent direction well and converges within a noticeably small number of iterations. For example, it takes about 10 iterations to converge in contrast to other algorithms, which take 40-200 iterations for convergence. Although there is a tradeoff between computational cost and the number of iterations, Modified Newton's method performs the best in most aspects (i.e., acceptably small f^* value universally compared to other algorithms, the number of iterations until convergence, total cost) since it converges in a few iterations.

However, a reasonable dimensionality of the objective problem should be presumed since its computational cost per iteration might dramatically increase if the dimension size or condition numbers exceeds around ten thousand or one-million levels. Specifically, regarding the tradeoff between cost per iteration and total iterations, Quasi-Newton methods such as BFGS are expected to show good performance among the ten algorithms we have tested. Contrary to DFP, which is more prone to divergence and unstable in convergence, BFGS behaves in a more stable manner, which can be expected to have good performance even in very large dimensions.

3.2.2 Experience and Recommendation

Regarding optimization algorithms, Gradient Descent is an intuitive method with a theoretical guarantee of convergence. However, it is often not recommended due to its sublinear convergence rate. Alternatively, Newton's Method is a powerful algorithm with a quadratic local convergence guarantee under convexity. The Modified Newton variation can be challenging to implement as it requires a Cholesky decomposition subroutine. BFGS and DFP are both quasi-Newton methods that are relatively easy to understand and implement. One major advantage of BFGS is that it does not require direct computation of the Hessian matrix. However, while DFP shares many similarities with BFGS, it may sometimes exhibit poor performance.

Based on our experience, Trust Region methods are not a preferred choice due to their complexity in coding and debugging, involving the use of CG subroutine, which may cause difficulty in execution and constructing codes.

4 Investigation of Initial Hessian Approximation in BFGS algorithm

We investigate if the choice of the initial hessian approximation has an impact on the performance of the BFGS algorithm. We consider 4 types of initial Hessian approximation, which are $I, 0.1I, 10I$, and the approach mentioned in 'Numerical Optimization' by Nocedal and Wright. Quoted from the textbook, it says that using a multiple of Identity matrix, βI as the initial Hessian is often vague in setting the scalar β, and the choice of β significantly affects the convergence. The last approach scales the starting matrix after the first step has been computed, but before the first BFGS update is performed. This initial

Hessian approximation is $H_0 \longleftarrow \frac{y_k^T s_k}{y_k^T y_k} I$. This formula is said to make the size of H_0 similar to that of $\nabla^2 f(x_0)^{-1}$. Hence, we investigate how the performance varies between different choices of initial Hessian for different problems.

According to the result, for convex quadratic problems, the initial Hessian with $0.1I$ shows the slowest convergence rate; it shows a linear convergence rate. In general, the Hessian update suggested by 'Numerical Optimization' shows the fastest convergence relative to the other Hessian options. In particular, the iterations are reduced by at least a factor of 3 for problem 7 (Rosenbrock).

As demonstrated in the convergence plots in **Appendix B**, overall, we find that $10I$ and the initial Hessian approximation from the textbook mostly show the best performances in terms of convergence rate. As mentioned above and from the behaviors of four different initial Hessian cases, we can conclude that using $0.1I$ as the initial Hessian seems it does not work well, while the other three cases can work effectively depending on which problem one is optimizing.

	Initial Hessian (H_0)			
Prob #	$H_0 = I$	$H_0 = 0.1I$	$H_0 = 10I$	approx. in textbook
1	-2.47088e+01	-2.47088e+01	-2.47088e+01	-2.47088e+01
2	-6.73629e+02	-6.73629e+02	-6.73629e+02	-6.73629e+02
3	-2.01497e+03	-2.01497e+03	-2.01497e+03	-2.01497e+03
4	-7.14635e+04	-7.14635e+04	-7.14635e+04	-7.14635e+04
5	1.31539e-19	8.14158e-15	5.12216e-16	2.29889e-17
6	1.53069e-03	1.01681e-03	1.81666e-03	1.02966e-03
7	7.67163e-12	5.57906e-12	6.11999e-11	1.45245e-09
8	3.98662e+00	3.98662e+00	3.98662e+00	3.98662e+00
9	9.31303e-15	4.48349e-14	6.00626e-12	2.61960e-13
10	-2.05573e-01	-2.05573e-01	-2.05573e-01	-2.05573e-01
11	-2.05573e-01	-2.05573e-01	-2.05573e-01	-2.05573e-01
12	8.94316e-09	5.19375e-09	7.31196e-08	9.33158e-10

Table 7: f^* value at termination (optimum if converges)

Regarding the optimal f value, BFGS with all initial Hessian conditions converge with a very small value, and there are no remarkable differences or improvements in minimizing the objective function value at convergence.

	Initial Hessian (H_0)			
Prob #	$H_0 = I$	$H_0 = 0.1I$	$H_0 = 10I$	*approx. in textbook*
1	1.751e-06	8.294e-07	2.159e-07	2.417e-07
2	4.605e-08	2.388e-08	1.180e-06	3.213e-06
3	9.841e-06	8.573e-06	9.460e-06	3.838e-06
4	9.900e-06	1.075e-05	5.121e-06	9.302e-06
5	4.217e-10	9.170e-08	2.408e-08	5.576e-09
6	3.174e-01	2.120e-01	3.207e-01	2.727e-01
7	7.077e-05	8.412e-05	1.036e-04	6.859e-05
8	1.849e-04	1.539e-04	6.984e-06	1.294e-04
9	1.016e-07	9.376e-07	2.269e-06	2.523e-06
10	8.966e-07	1.799e-06	4.893e-08	1.081e-06
11	8.107e-07	2.037e-06	2.671e-06	6.015e-09
12	3.215e-05	1.896e-05	9.840e-05	1.006e-05

Table 8: $\|\nabla f\|$ at termination (optimum if converges)

	Initial Hessian (H_0)			
Prob #	$H_0 = I$	$H_0 = 0.1I$	$H_0 = 10I$	approx. in textbook
1	25	35	14	22
2	44	58	29	42
3	31	85	59	27
4	323	842	224	287
5	3	10	4	3
6	17	20	19	14
7	31	32	26	33
8	104	70	108	22
9	14	13	16	12
10	6	11	20	14
11	6	23	24	7
12	31	27	41	25

Table 9: Number of iterations

In terms of the number of iterations until it reaches its convergence, $H_0 = 10I$ and *approximation in the textbook* show the best performance in having the small number of iterations. Problems 3 and 4, which have specifically high dimension sizes, are expected to take more iterations to converge than other algorithms. To be specific, in these problems, $H_0 = 10I$ converges within fewer iterations and apparently performs better than other conditions.

	Initial Hessian (H_0)			
Prob #	$H_0 = I$	$H_0 = 0.1I$	$H_0 = 10I$	approx. in textbook
1	0.044	0.057	0.011	0.035
2	0.084	0.102	0.047	0.083
3	14.711	38.830	13.535	10.375
4	166.724	415.165	55.869	158.737
5	0.001	0.002	0.001	0.000
6	0.004	0.004	0.005	0.003
7	0.003	0.004	0.003	0.004
8	0.218	0.114	0.261	0.036
9	0.003	0.002	0.003	0.002
10	0.002	0.003	0.005	0.003
11	0.452	1.809	1.835	0.614
12	0.008	0.007	0.012	0.007

Table 10: Computational Cost (time in seconds)

As described in previous sections of comparing the number of iterations, in problems 3 and 4, $10I$ also performs the best, which corresponds to the result from the number of iterations. We empirically found that $H_0 = 10I$ works the most efficiently when using the BFGS algorithm in higher dimension problems since its iterations and computational cost are both the smallest, which means its total computation time and complexity are the least for high dimensional problems. However, in normal ranges, all three conditions except for $H_0 = 0.1I$ have a similar level of performance that $10I$ is only distinguishably efficient in some extreme problem settings as explained above.

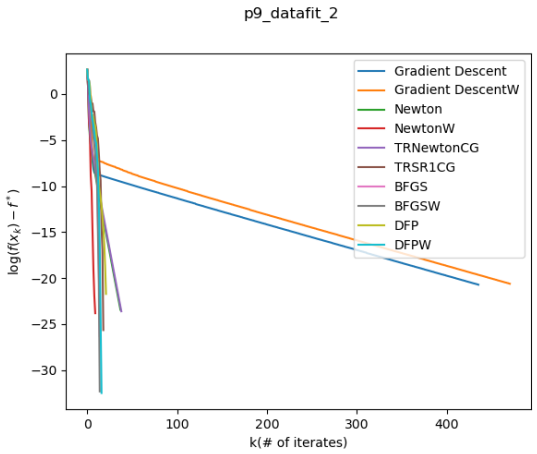
5 Conclusion

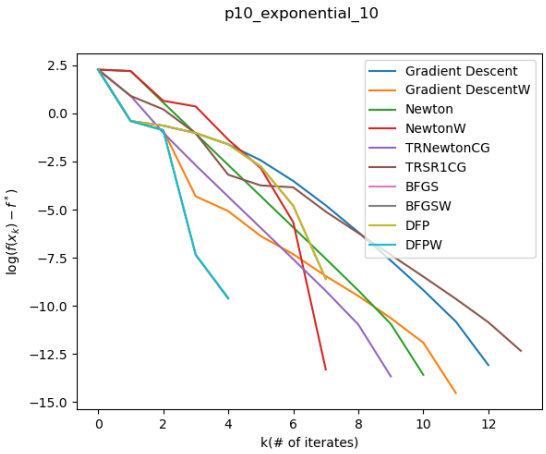
In short, we identify *Modified Newton with Wolfe Line Search* as the state-of-the-art algorithm under the capability of computing the exact Hessian matrix with a reasonable dimension size of the objective

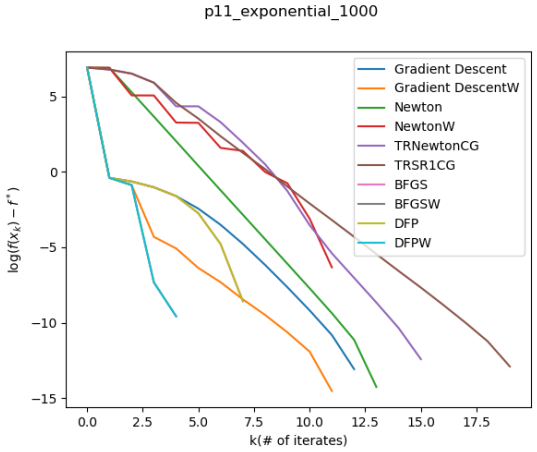
function. We also conclude that the performance of the BFGS algorithm is not very sensitive to the choice of the initial Hessian approximation; however, $H_0 = 0.1I$ is not recommended for use due to its poor performance in convergence.

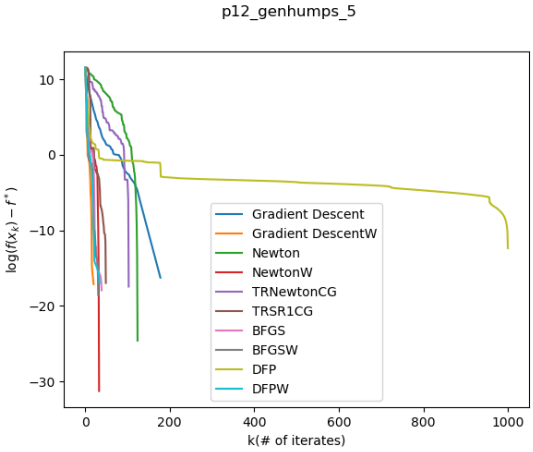
Appendix A

Figures from Section 3



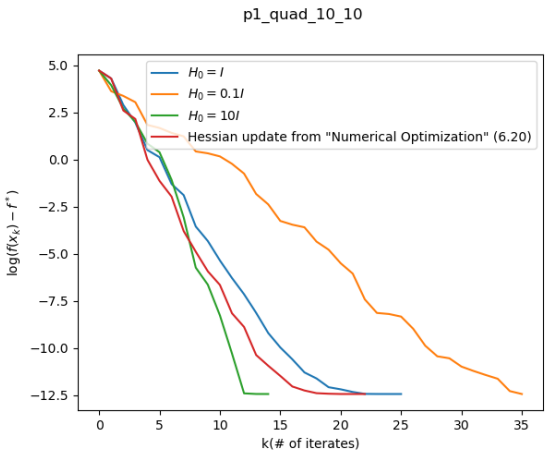


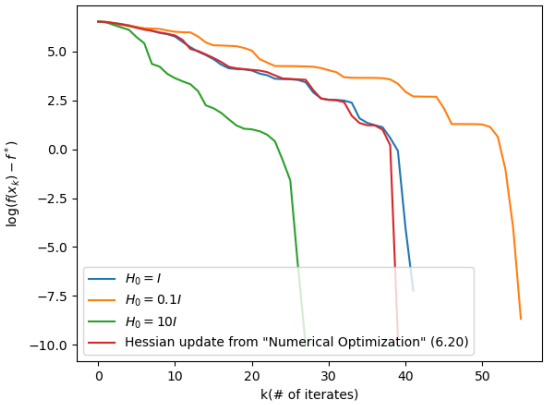




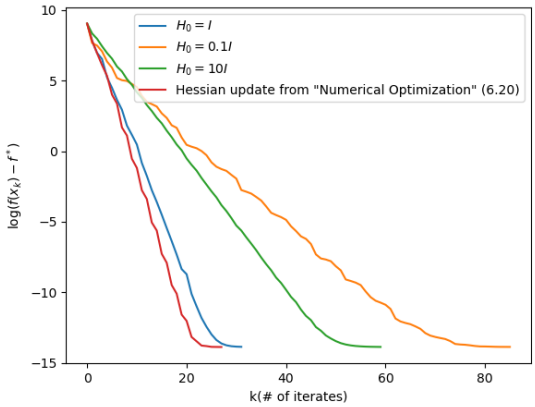
Appendix B

Figures from Section 4

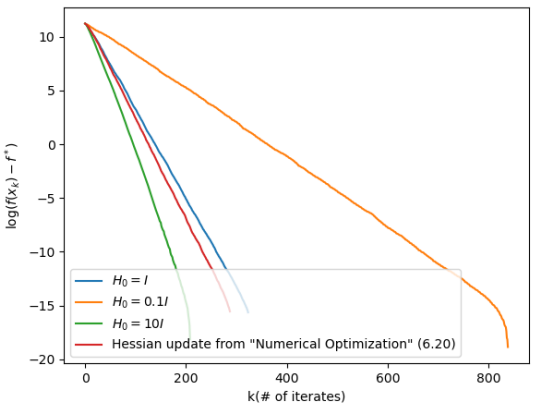




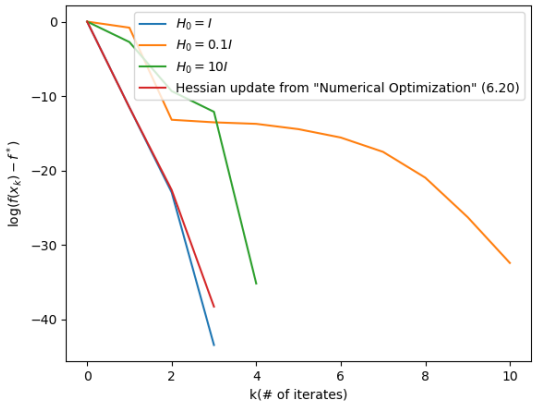
p3_quad_1000_10



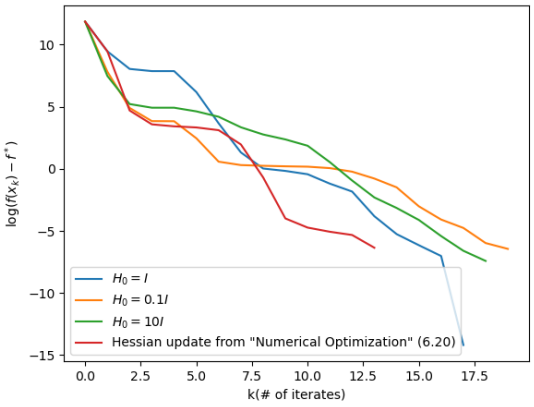
p4_quad_1000_1000



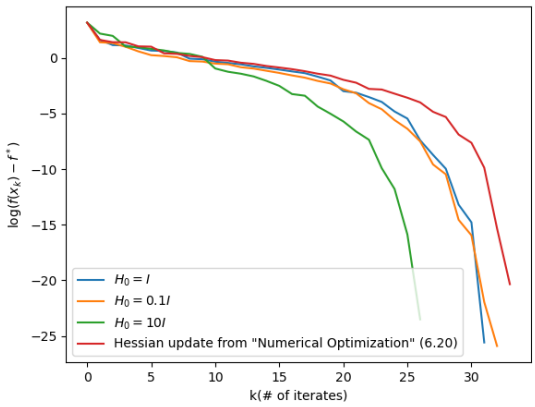
p5_quartic_1



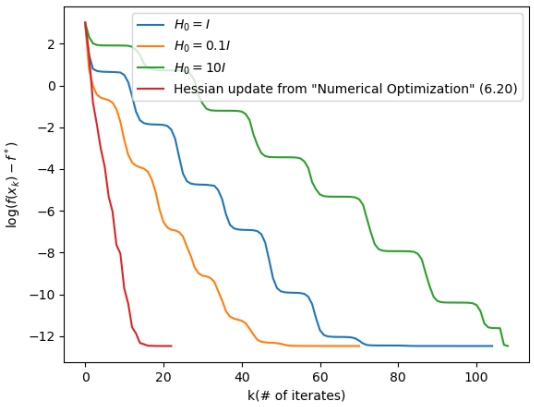
p6_quartic_2



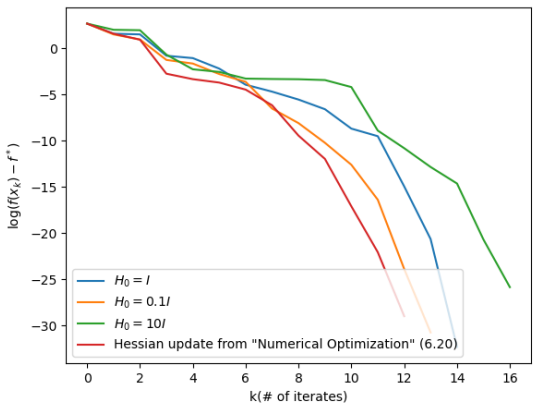
p7_rosenbrock_2



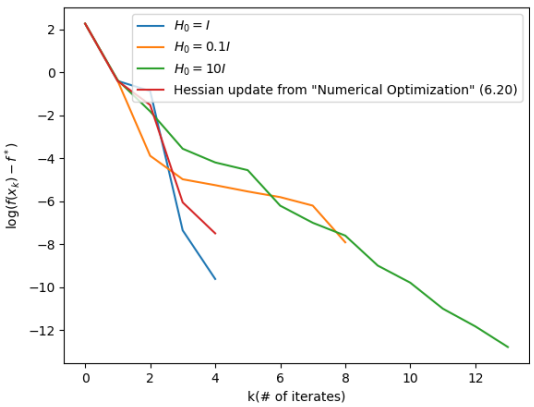
p8_rosenbrock_100

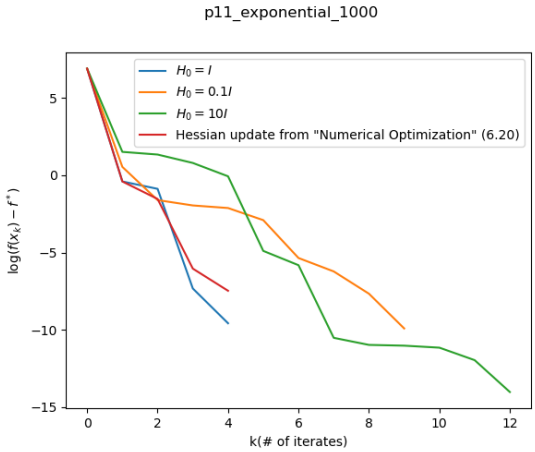


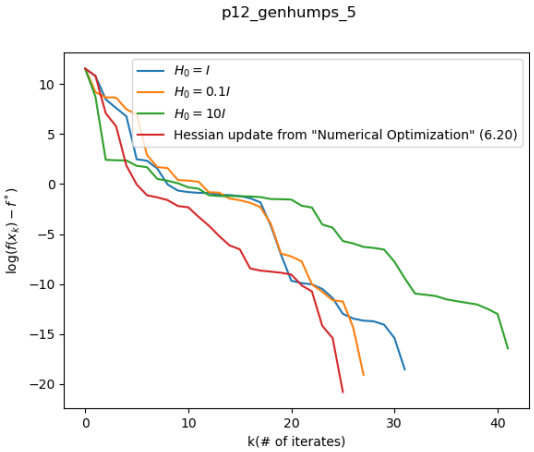
p9_datafit_2



p10_exponential_10




p11_exponential_1000
$H_0 = I$
$H_0 = 0.1I$
$H_0 = 10I$
Hessian update from "Numerical Optimization" (6.20)
$\log(f(x_k) - f^*)$
k(# of iterates)


p12_genhumps_5
$H_0 = I$
$H_0 = 0.1I$
$H_0 = 10I$
Hessian update from "Numerical Optimization" (6.20)
$\log(f(x_k) - f^*)$
k(# of iterates)