2026
IOE 511 Project
Summary Sheet

Group Members
Jincheng Ye
Yukai Wang
Yuwei Zhuo

Tuning the Step: How Wolfe Parameters Shape Optimization Performance

Summary

This project investigates the performance of a range of classical continuous optimization algorithms, including first-order methods, second-order methods, quasi-Newton methods, and trust-region approaches, on a standardized set of twelve test problems. The primary goal is to compare their convergence behavior, efficiency, and robustness across problems with different characteristics, such as conditioning, dimensionality, and nonconvexity.

We first implement ten algorithms, including Gradient Descent (with backtracking and Wolfe line search), modified Newton methods, trust-region methods (Newton-CG and SR1-CG), and quasi-Newton methods (BFGS and DFP, with both backtracking and Wolfe line search). A unified set of default parameters is selected based on empirical tuning to ensure a fair and stable comparison across all problems.

To evaluate baseline performance, we analyze multiple metrics including iteration counts, function evaluations, gradient evaluations, CPU time, and convergence profiles in terms of optimality gaps. The results show clear distinctions between algorithm classes: second-order and trust-region methods achieve rapid convergence and are able to reach extremely small optimality gaps, while first-order methods converge more slowly and often fail to achieve high accuracy within a limited number of iterations. Quasi-Newton methods demonstrate intermediate performance, balancing efficiency and robustness but showing sensitivity to problem conditioning.

In addition to baseline comparisons, we investigate a focused research question on the sensitivity of algorithm performance to Wolfe line search parameters (c_1, c_2). By systematically evaluating a wide range of parameter combinations across multiple algorithms and problems, we assess both average performance and robustness using performance profiles. The analysis reveals that no single parameter choice is universally optimal, but certain configurations consistently provide strong performance across different settings. Based on these findings, we identify practical parameter choices for each algorithm that balance convergence speed and computational cost.

Overall, the study highlights the trade-offs between algorithm complexity, convergence speed, and robustness, and demonstrates the importance of both problem structure and parameter selection in determining optimization performance.

Keywords: Continuous Optimization, Gradient Descent, Newton, Quasi Newton, Trust Region, Wolfe Line Search, Parameter Adjust

Contents

1 Algorithms

1.1 Algorithms Overview

Gradient Descent with Backtracking Line Search. This is a line search method that uses the steepest descent direction $d_k = -\nabla f(x_k)$. The step size is determined via backtracking line search, enforcing the Armijo condition. The method is globally convergent under standard assumptions, but typically exhibits at best linear convergence and may perform poorly in ill-conditioned regions.

Gradient Descent with Wolfe Line Search. This method also uses the steepest descent direction, but the step size satisfies the Wolfe conditions, incorporating both sufficient decrease and curvature requirements. Compared to backtracking, it often yields more appropriate step sizes and better practical efficiency. However, it still relies only on first-order information and does not exploit curvature in nonconvex problems.

Modified Newton Method with Backtracking Line Search. This is a line search method where the step is computed using second-order information: $d_k = -H_k^{-1}\nabla f(x_k)$. The Hessian is modified when necessary to ensure positive definiteness and guarantee a descent direction. Backtracking line search determines the step size. The method can achieve quadratic local convergence when the Hessian modification becomes inactive near the solution, while in nonconvex regions, Hessian modification improves stability.

Modified Newton Method with Wolfe Line Search. This variant combines the modified Newton direction with Wolfe line search, improving step size selection and overall robustness. It can retain fast, potentially quadratic, local convergence when the Hessian modification becomes inactive near the solution. Nonconvexity is handled through Hessian modification, while Wolfe conditions enhance efficiency.

Trust Region Newton-CG. This is a trust region method in which each step is obtained by approximately solving a quadratic subproblem within a region $\|d\| \leq \Delta$. The subproblem is solved approximately using truncated CG, which can terminate early when negative curvature is encountered. This makes the method suitable for large-scale problems. Nonconvexity is handled naturally by the trust region constraint and the ability to detect negative curvature.

Trust Region SR1-CG. This method also follows a trust region framework but uses an SR1 quasi-Newton approximation of the Hessian. Unlike BFGS, SR1 allows the Hessian approximation to be indefinite, enabling better modeling of nonconvex curvature. The trust region subproblem is solved approximately using truncated CG, providing robustness and the ability to exploit negative curvature directions.

BFGS with Backtracking Line Search. This is a quasi-Newton line search method that builds a positive definite approximation to the Hessian. The search direction is $d_k = -H_k^{-1}\nabla f(x_k)$, and the step size is determined via backtracking. It often achieves superlinear local convergence in practice. However, by enforcing positive definiteness, it cannot exploit negative curvature in nonconvex regions.

BFGS with Wolfe Line Search. This variant combines BFGS with Wolfe line search, which helps ensure the curvature condition $s_k^\top y_k > 0$ required for stable updates. It is generally more robust and efficient than the backtracking version. As with standard BFGS, it does not exploit negative curvature.

DFP with Backtracking Line Search. This is a quasi-Newton line search method using the DFP update to approximate the inverse Hessian. While structurally similar to BFGS, it is typically less stable and more sensitive to numerical issues. Backtracking line search is used for step selection. Like BFGS, it preserves positive definiteness when the curvature condition holds and cannot explicitly handle negative curvature.

DFP with Wolfe Line Search. This method combines the DFP update with Wolfe line search, improving step size selection and helping enforce the curvature condition required for valid updates. Although more stable than the backtracking version, it generally performs worse than BFGS. Its handling of nonconvexity is limited by the enforced positive definiteness of the Hessian approximation.

1.2 Default Algorithm Parameters

Based on empirical testing across all problems, the following default parameter values were selected to balance robustness and efficiency. The optimality tolerance and maximum iteration limit are fixed as required, while the remaining parameters are tuned for stable performance across algorithms.

Parameter	Symbol	Default Value
Optimality tolerance	`term_tol`	1×10^{-6}
Maximum iterations	`max_iter`	1×10^{3}
Initial step size (line search)	α_0	1.0
Backtracking reduction factor	ρ	0.5
Armijo parameter	c_1	1×10^{-4}
Wolfe curvature parameter	c_2	0.5
Initial trust region radius	Δ_0	1.0
Maximum trust region radius	$\Delta_{\max}$	100.0
Trust region acceptance threshold	η	0.1
SR1 safeguard threshold	ϵ_{SR1}	1×10^{-8}
BFGS/DFP curvature threshold	ϵ_{sy}	1×10^{-6}
L-BFGS memory size	m	10

Table 1: Default parameter settings used across all algorithms.

2 Baseline Performance

After implementing the 10 algorithms across 12 problems, we recorded the performance information including iterations, function evaluations, gradient evaluations, and CPU running time, as shown in

problem	BFGS	BFGSW	DFP	DFPW	GradientDescent	GradientDescentW	Newton	NewtonW	TRNewtonCG	TRSR1CG
P10_exponential_10	19	23	1000	104	27	27	13	13	12	17
P11_exponential_100	10	6	10	6	21	18	13	13	13	18
P12_genhumps_5	57	33	1000	1000	148	166	109	45	125	141
P1_quad_10_10	17	17	16	16	50	50	1	1	4	10
P2_quad_10_1000	13	13	13	13	1000	1000	1	1	3	13
P3_quad_1000_10	25	25	31	31	51	51	1	1	7	26
P4_quad_1000_1000	384	384	275	275	1000	1000	1	1	7	265
P5_quartic_1e-4	3	3	3	3	2	2	2	2	3	3
P6_quartic_1e4	28	28	18	18	5	5	12	12	12	27
P7_rosenbrock_2	33	33	47	35	1000	1000	20	20	23	109
P8_rosenbrock_100	112	112	108	108	42	42	4	4	4	42
P9_datafit_2	14	14	21	21	530	530	8	8	7	19

(a) Number of iterations

problem	BFGS	BFGSW	DFP	DFPW	GradientDescent	GradientDescentW	Newton	NewtonW	TRNewtonCG	TRSR1CG
P10_exponential_10	22	28	1006	224	42	42	31	31	13	18
P11_exponential_100	14	14	14	14	24	23	31	31	14	19
P12_genhumps_5	81	65	1051	1359	226	382	143	102	126	142
P1_quad_10_10	26	26	20	20	168	168	2	2	5	11
P2_quad_10_1000	59	59	44	44	9959	9959	2	2	4	14
P3_quad_1000_10	76	76	64	64	171	171	2	2	8	27
P4_quad_1000_1000	2976	2976	1389	1389	9958	9958	2	2	8	266
P5_quartic_1e-4	4	4	4	4	3	3	3	3	4	4
P6_quartic_1e4	97	97	86	86	73	73	13	13	13	28
P7_rosenbrock_2	53	53	67	57	9834	9883	28	28	24	110
P8_rosenbrock_100	1012	1012	614	614	462	462	5	5	5	43
P9_datafit_2	23	23	30	30	2908	2908	16	16	8	20

(b) Number of function evaluations

problem	BFGS	BFGSW	DFP	DFPW	GradientDescent	GradientDescentW	Newton	NewtonW	TRNewtonCG	TRSR1CG
P10_exponential_10	23	29	1007	225	43	43	32	32	14	19
P11_exponential_100	15	15	15	15	25	24	32	32	15	20
P12_genhumps_5	82	66	1052	1360	227	383	144	103	100	110
P1_quad_10_10	27	27	21	21	169	169	3	3	6	12
P2_quad_10_1000	60	60	45	45	9960	9960	3	3	5	12
P3_quad_1000_10	77	77	65	65	172	172	3	3	9	28
P4_quad_1000_1000	2977	2977	1390	1390	9959	9959	3	3	9	235
P5_quartic_1e-4	5	5	5	5	4	4	4	4	5	5
P6_quartic_1e4	98	98	87	87	74	74	14	14	14	26
P7_rosenbrock_2	54	54	68	58	9835	9884	29	29	22	81
P8_rosenbrock_100	1013	1013	615	615	463	463	6	6	6	25
P9_datafit_2	24	24	31	31	2909	2909	17	17	9	18

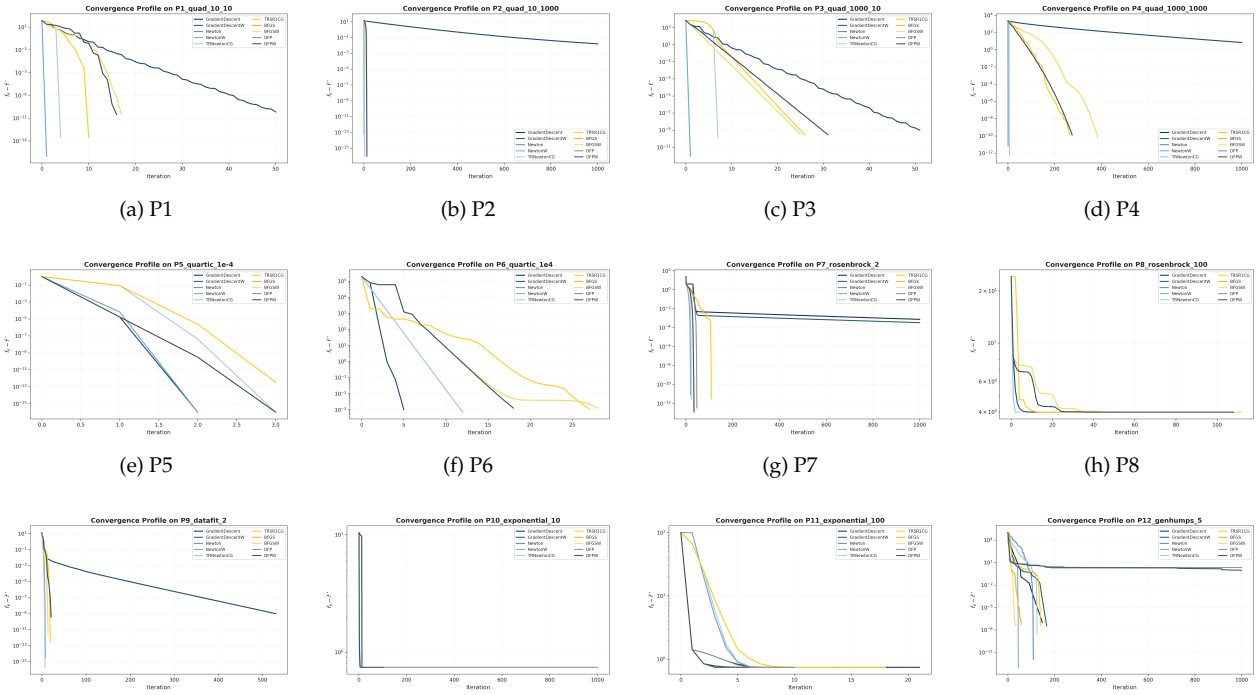
(c) Number of gradient evaluations

problem	BFGS	BFGSW	DFP	DFPW	GradientDescent	GradientDescentW	Newton	NewtonW	TRNewtonCG	TRSR1CG
P10_exponential_10	0.000833	0.001012	0.025007	0.005075	0.000903	0.000817	0.007877	0.000962	0.000902	0.002126
P11_exponential_100	0.001065	0.000822	0.000790	0.000657	0.000532	0.000489	0.002902	0.003041	0.001376	0.001927
P12_genhumps_5	0.002483	0.001915	0.031849	0.054616	0.004793	0.008020	0.011176	0.005301	0.007316	0.007851
P1_quad_10_10	0.000804	0.001096	0.000717	0.000640	0.258659	0.003052	0.000436	0.000193	0.000422	0.000925
P2_quad_10_1000	0.002001	0.001201	0.000847	0.001661	0.161472	0.321474	0.000207	0.000101	0.000402	0.001154
P3_quad_1000_10	1.149473	1.578324	0.254844	0.281622	0.072173	0.036589	0.032628	0.024601	0.022068	0.202388
P4_quad_1000_1000	17.314919	11.581386	1.403016	1.490065	2.649812	2.138507	0.030361	0.023518	0.050776	1.819907
P5_quartic_1e-4	0.000277	0.000279	0.000169	0.000168	0.000321	0.000151	0.001648	0.000956	0.000378	0.000555
P6_quartic_1e4	0.002360	0.002964	0.001944	0.001983	0.001836	0.001887	0.000803	0.000708	0.001031	0.002101
P7_rosenbrock_2	0.002190	0.003065	0.002895	0.002050	0.139785	0.196703	0.001710	0.001473	0.001652	0.007804
P8_rosenbrock_100	0.137037	0.137719	0.140660	0.110163	0.077877	0.064558	0.001924	0.001665	0.002489	0.007323
P9_datafit_2	0.000554	0.000561	0.000710	0.000728	0.037903	0.039647	0.000993	0.000634	0.000439	0.000911

(d) CPU time in seconds

Figure 1: Summary of results

Fig.2 shows the optimality gaps of all the 10 algorithms on each of the 12 problems. Across the 12 test problems, clear differences can be observed in both convergence speed and the final attainable optimality



(a) P1 (b) P2 (c) P3 (d) P4

(e) P5 (f) P6 (g) P7 (h) P8

(i) P9 (j) P10 (k) P11 (l) P12

Figure 2: Convergence profiles for the 12 test problems.

gap among the ten algorithms. First, Newton-type methods (Newton, NewtonW, and TRNewtonCG) consistently exhibit the fastest convergence. On most problems (e.g., P1–P4, P7, P9), they reduce the optimality gap to extremely small magnitudes (often below 10^{-12}) within only a few iterations, demonstrating their strong local quadratic convergence behavior. Similarly, trust-region and quasi-Newton methods such as TRSR1CG, BFGS, and BFGSW also achieve very small optimality gaps (typically 10^{-8} to 10^{-12}) but generally require more iterations than Newton-type methods.

In contrast, first-order methods (GradientDescent and GradientDescentW) converge much more slowly. Although they show steady and monotone decrease in the optimality gap, they often require hundreds or even thousands of iterations to reach moderate accuracy, and on several problems (e.g., P2, P4, P8), they fail to reduce the gap below relatively large levels compared to second-order methods. This highlights the well-known limitation of first-order methods on ill-conditioned or high-dimensional problems.

Quasi-Newton methods (BFGS, DFP, and their Wolfe variants) show intermediate behavior. BFGS and BFGSW are generally robust and converge to very small optimality gaps on most problems, although their convergence speed can degrade on ill-conditioned settings (e.g., P4, P8). DFP and DFPW are less stable. While they converge on simpler problems, they sometimes plateau at higher optimality gaps or show slower progress compared to BFGS, indicating sensitivity to curvature approximation quality.

Several problem specific phenomena are also worth noting. On the Rosenbrock problems (P7 and P8), most methods experience difficulty due to the narrow curved valley. Only second-order and trust-region methods achieve very small optimality gaps, while first-order methods stagnate at relatively high levels. On the exponential problems (P10 and P11), all methods converge rapidly, indicating that these problems are relatively well-conditioned. However, on the nonconvex Genhumps problem (P12), many algorithms including quasi-Newton methods stagnate at relatively large optimality gaps (around 10^{-1}), suggesting convergence to local minima rather than the global optimum. Finally, for some quartic problems (P5 and P6), most methods successfully reach extremely small optimality gaps, but trust-region methods tend to require more iterations compared to Newton-type methods.

Overall, the results highlight that second-order and trust-region methods are significantly more effective at achieving very small optimality gaps, while first-order methods may fail to converge to high accuracy within a reasonable number of iterations. Additionally, problem structure (conditioning and nonconvexity)

plays a critical role in determining whether an algorithm can converge to near-optimal solutions.

3 Comments and Algorithm of Choice

3.1 Comments on the Results

No single algorithm dominates across all twelve problems, but performance differences can be explained by specific iteration counts together with the structural features of each problem.

On the four quadratic problems (P1–P4), Newton and NewtonW converge in a *single* iteration, while BFGS requires 345 iterations on P4 (`quad_1000_1000`) and GradientDescent hits the 1000-iteration cap on P2, P4, and P7. On the nonconvex Genhumps problem (P12), NewtonW converges in 39 iterations and TRNewtonCG in 70, whereas DFP reaches the 1000-iteration limit on both P10 and P12. These gaps cannot be attributed to implementation details alone and instead reflect three structural factors discussed below.

Structural explanations:

1. **Conditioning.** First-order methods and DFP rely on limited or gradually accumulated curvature information, so large condition numbers severely degrade their performance. Newton-type methods use the exact Hessian and are much less sensitive to ill-conditioning, which explains the order-of-magnitude gap in iteration counts on P4.
2. **Indefinite curvature.** BFGS and DFP enforce positive-definite Hessian approximations and therefore cannot represent negative-curvature directions of the true Hessian, causing slowdown or stagnation on problems with indefinite curvature such as the Rosenbrock problems (P7, P8) and Genhumps (P12). Trust-region methods (TRNewtonCG, TRSR1CG) handle indefiniteness naturally through truncated CG steps combined with the trust-region constraint, which is why they remain reliable on P12.
3. **Update-formula stability.** Independently of conditioning and curvature, DFP updates are numerically less robust than BFGS updates. On the exponential problem P10 – which is relatively well-conditioned and convex – DFP still reaches the 1000-iteration limit while BFGS converges without difficulty. This isolates the difference to the update formula itself rather than to problem structure.

The quartic problem with the large coefficient (P6) is difficult for *all* tested methods: every algorithm terminates with a relatively large gradient norm under our stopping criterion. This indicates that the challenge originates from the scaling and structure of the problem itself rather than from a weakness of any individual algorithm.

3.2 Algorithm of Choice

For practical use, we select **NewtonW** (Modified Newton with Wolfe line search) as our algorithm of choice. This decision balances three key considerations: convergence speed, ease of implementation, and the results of our parameter study.

First, NewtonW performs very well on most problems, especially the Rosenbrock problems (P7, P8), where it requires only 20 and 4 iterations, respectively, compared with 33 and 112 iterations for BFGSW. It solves all four quadratic problems in a single iteration and converges rapidly on data-fitting and exponential problems.

Second, we have rigorously optimized its Wolfe line-search parameters through our Big Question study. The extensive parameter sweep across 90 combinations of (c_1, c_2) pairs revealed that NewtonW performs best with $(c_1, c_2) = (0.05, 0.5)$, a choice further supported by performance profiles showing strong robustness across problems. This close connection between our algorithm selection and our parameter study makes NewtonW a principled choice that directly reflects our investigation.

Third, NewtonW requires explicit Hessian computation, but in our setting it avoids the added complexity of trust-region radius updates and inner CG iterations, making it simpler to implement and tune than trust-region Newton methods. While TRNewtonCG appears more robust on the most nonconvex problem

in our test set (P12), that benefit comes at the cost of additional parameters (trust region radius scaling γ_1, γ_2; CG tolerance and iteration limits) that fall outside the scope of our Big Question investigation.

For small-to-medium scale problems where second derivatives are available, NewtonW with parameters $(c_1, c_2) = (0.05, 0.5)$ and $\alpha_0 = 1.0$ offers a compelling balance of speed, robustness to moderate nonconvexity, and ease of implementation.

3.3 Verification on Rosenbrock

To validate our algorithm of choice, we run NewtonW with its optimal Wolfe parameters $(c_1, c_2) = (0.05, 0.5)$ on both Rosenbrock problems. Table 2 reports the results.

Problem	Iter	f-eval	g-eval	CPU (s)	$\|\nabla f\|_\infty$
P7 Rosenbrock (n=2)	20	26	27	0.002	8.3×10^{-7}
P8 Rosenbrock (n=100)	4	5	6	0.005	6.5×10^{-7}

Table 2: NewtonW with optimal parameters on Rosenbrock.

Notably, the tuned parameters outperform the default $(c_1, c_2) = (10^{-4}, 0.9)$ from the baseline: on P7, the final objective improves from 8.5×10^{-12} to 2.8×10^{-13}, confirming the practical value of the parameter study from our Big Question.

4 Big Question

In this section, a specific aspect of these algorithms is investigated: **How sensitive is algorithm performance to the choice of Wolfe line search parameters, and what are, on average, the best choices for the parameters?** The goal of this section is to evaluate the sensitivity of algorithm performance to (c1, c2) and to identify parameter choices that perform well on average across problems.

To address this problem, we selected the four algorithms in our algorithm library which use Wolfe line search: GradientDescentW, NewtonW, BFGSW, and DFPW. Next, the report will go through the research steps of solving the problem: (i) decide the range of value we will investigate for (c_1, c_2), (ii) performance metries and average performance evaluation, (iii) performance profiles and best parameters choices, and (iv) sensitivity analysis.

4.1 range of value of (c_1, c_2)

As the Wolfe line search conditions show:

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^\top d_k,$$

$$f(x_k + \alpha_k d_k)^\top d_k \geq c_2 \nabla f(x_k)^\top d_k.$$

The parameter c_1 governs the sufficient decrease (Armijo) condition, which ensures that the step size α_k produces a meaningful reduction in the objective function. A smaller value of c_1 relaxes this condition, allowing steps to be accepted more easily, potentially leading to larger step sizes and faster progress. However, if c_1 is too large, the condition becomes overly restrictive, resulting in excessively small step sizes and slow convergence.

The parameter c_2 controls the curvature condition, which enforces that the directional derivative at the new point is sufficiently reduced relative to the initial slope. This condition prevents premature acceptance of steps that do not adequately reduce the gradient along the search direction. Larger values of c_2 make the condition less restrictive, allowing more aggressive steps, whereas smaller values impose stricter curvature

requirements, potentially leading to conservative step sizes and increased computational effort.

Theoretical guidelines require that $0 < c_1 < c_2 < 1$. First, $c_1 << 1$ ensures that the sufficient decrease condition is not overly restrictive. If c_1 were too large, many potentially useful steps would be rejected, leading to excessively small step sizes and slow convergence. On the other hand, selecting larger c_2 relaxes the curvature condition, allowing the directional derivative at the new point to remain relatively close to the initial slope. In particular, for quasi-Newton methods such as BFGS, a larger c_2 helps ensure that the curvature information used in Hessian approximation remains meaningful while still enabling sufficiently aggressive progress. Therefore, we chose to set the range of c_1 to be $\{10^{-5}, 5 \times 10^{-5}, 10^{-4}, 5 \times 10^{-4}, 0.001, 0.005, 0.01, 0.05, 0.099\}$, and the range of c_2 to be $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$.

Then, we applied the four algorithms with different (c_1, c_2) (with a total of $9 \times 10 = 90$ combinations) on the problem set.

Fitting all these 90 sets of parameters into the algorithms, we used heat maps to show the number of iterations needed to converge under each (c_1, c_2) combination. Here in Fig.3, we select 5 representative problems (quadratic problem with $n = 10$ and $\kappa = 10$, quartic problem_1, Rosenbrock problem with $n = 2$, datafit problem with $n = 2$, exponential problem with $n = 10$) to present a general idea of the how the change in (c_1, c_2) influences the performance of the algorithms. Each row represents an algorithm, and each column represents the same problem.

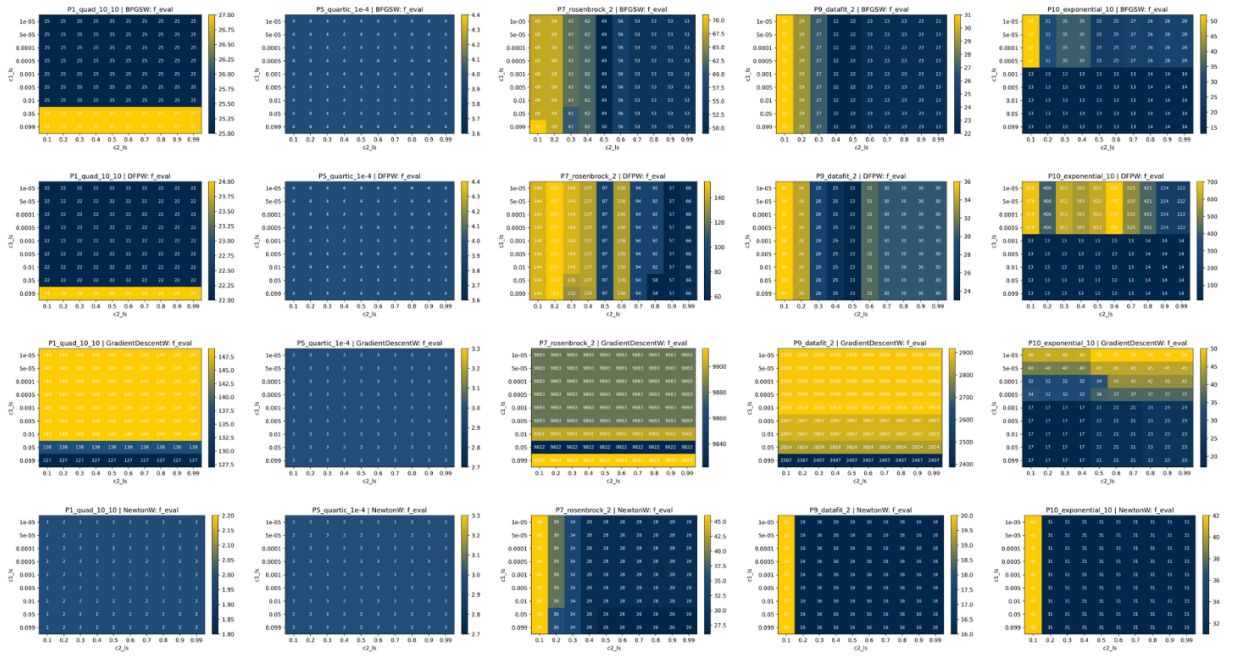


Figure 3: The overall pattern of algorithms performance on different questions. From top to bottom, each row represents: BFGS, DFP, GD, Newton. From left to right, each column represents: quad_10_10, quartic_1e-4, rosenbrock_2, datafit_2, exponential_10.

It is not difficult to observe that algorithm performance varies significantly across different problems. That is to say, there is not a single (c_1, c_2) that gives best performance for all questions. However, it is still possible to identify parameter choices that perform well on most instances.
To this end, we evaluate parameter performance using two complementary criteria: average performance and performance profiles.

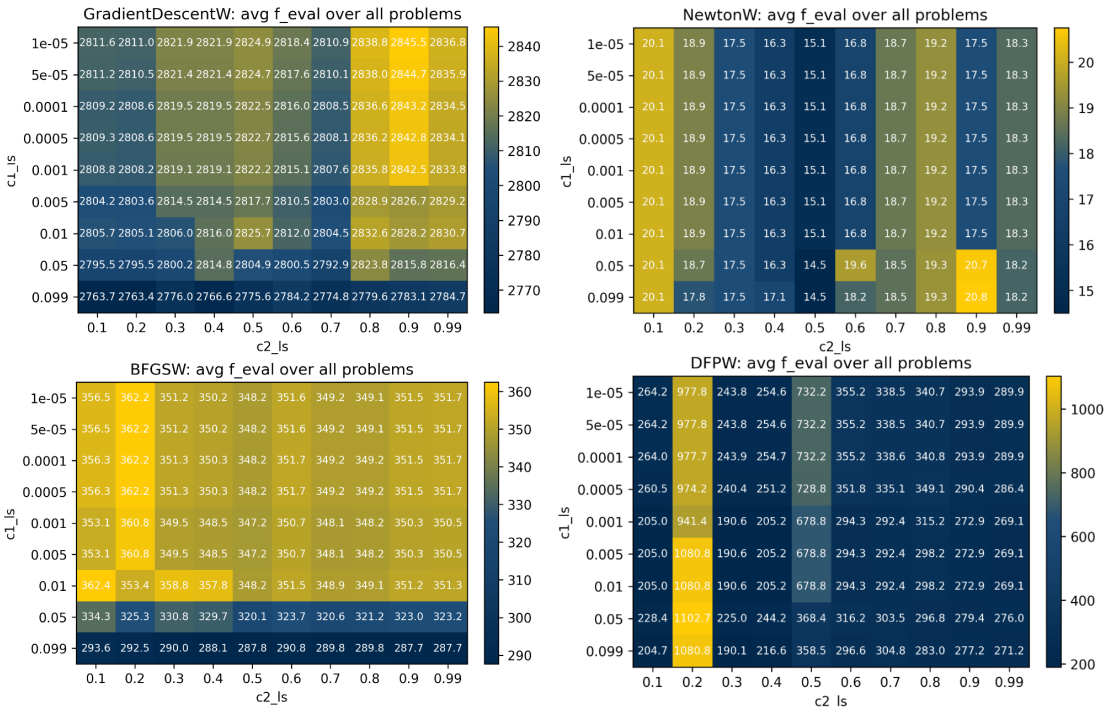


Figure 4: The heat maps of the four algorithms average performance over all the problems.

4.2 Average Performance

To account for variability across the 12 test problems, we compute average performance metrics, as shown in Fig. 4. The resulting heatmaps provide an overall view of how parameter choices influence performance. Based on these observations, we obtain the following preliminary conclusions:

1. For Gradient Descent, smaller values of c_1 consistently lead to better performance, and c_2 shows the same pattern. The best average performance is achieved when $(c_1, c_2) = (0.099, 0.1)$.

2. For Newton's method, the effect of c_1 depends on the choice of c_2: for some values of c_2, performance improves as c_1 decreases, while for others the trend is reversed. Overall, the best average performance is observed at $(c_1, c_2) = (0.05, 0.5)$.

3. For BFGS, smaller values of c_1 again lead to improved performance, while c_2 shows an opposite trend, with a few exceptions. The best average performance is achieved at $(c_1, c_2) = (0.099, 0.99)$ or $(0.099, 0.9)$.

4. For DFP, no clear monotonic pattern is observed. However, performance declines noticeably when $c_2 = 0.2$ or 0.5. The best average performance is obtained at $(c_1, c_2) = (0.099, 0.3)$.

While average performance provides a useful summary of overall efficiency, it may be influenced by outliers and does not fully capture the consistency of a parameter choice across problems.

4.3 Performance Profiles

To address this limitation, we further evaluate parameter choices using performance profiles. Compared to average performance, performance profiles emphasize robustness, as they capture how consistently a configuration performs close to the best across a range of problems.
For a given metric (e.g., function evaluations or CPU time), the performance profile of a parameter configuration measures the fraction of problems for which its performance is within a factor τ of the best observed

performance. Formally, it is defined as:

$$\rho(\tau) = \frac{1}{P}\{p \in P : \frac{t_p}{min_j t_{p,j}} \leq \tau\},$$

where t_p denotes the performance on problem p, and the denominator represents the best performance across all parameter choices.

We built our analysis towards performance profiles on the basis of average performance. To be specific, we investigated the cpu running time of the top 10 parameter pairs ranked by average performance. This restriction ensures that we focus on reasonably efficient configurations while avoiding clearly suboptimal choices. The results are shown in Fig.5,6,7,8. For DFP and Newton method, the parameter pairs with quickest convergence also has a smallest cpu running time, so the best (c_1, c_2) is easily determined. For Gradient Descent, $(0.099, 0.1)$ has a slightly slower convergence than $(0.099, 0.2)$, however, it has a much smaller cpu running time. Therefore, balancing the efficiency and cpu usage, we should choose $(0.099, 0.1)$ as the best choice. For BFGS, $(0.099, 0.9)$ and $(0.099, 0.99)$ share the same f_eval, but the fronter one has a smaller cpu running time, so we should choose $(0.099, 0.9)$.

In conclusion, the final best choices for the parameter pairs are:

BFGS	(0.099,0.9)
DFP	(0.099,0.3)
Gradient Descent	(0.099,0.1)
Newton	(0.05,0.5)

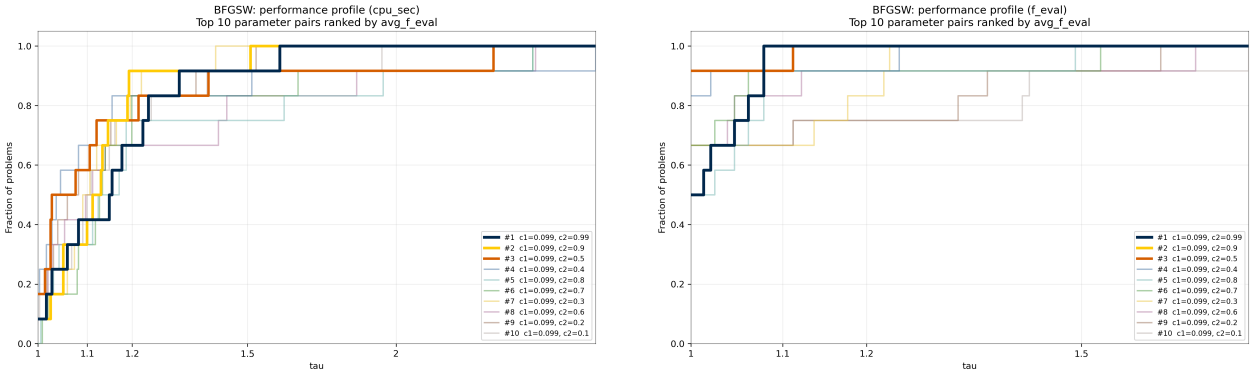


Figure 5: The cpu running time and iterations of BFGS with the top 10 parameter pairs ranked by average performance.

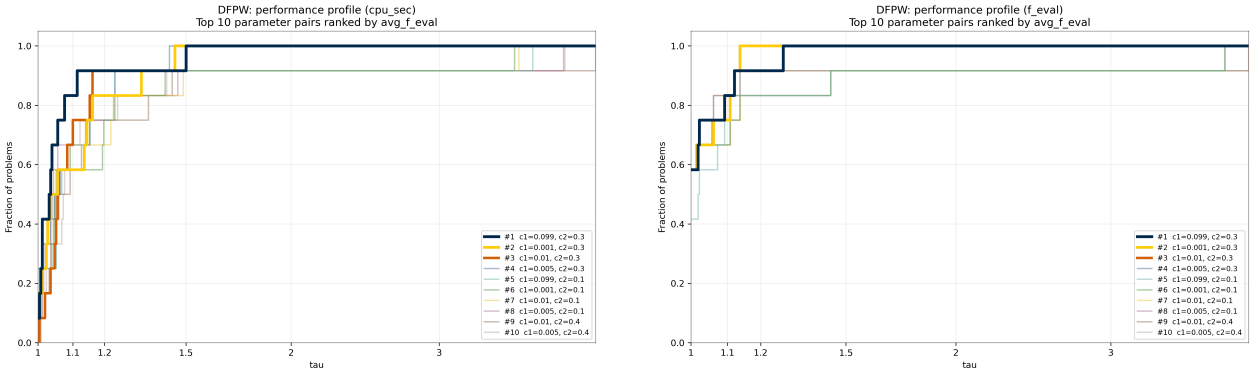


Figure 6: The cpu running time and iterations of DFP with the top 10 parameter pairs ranked by average performance.

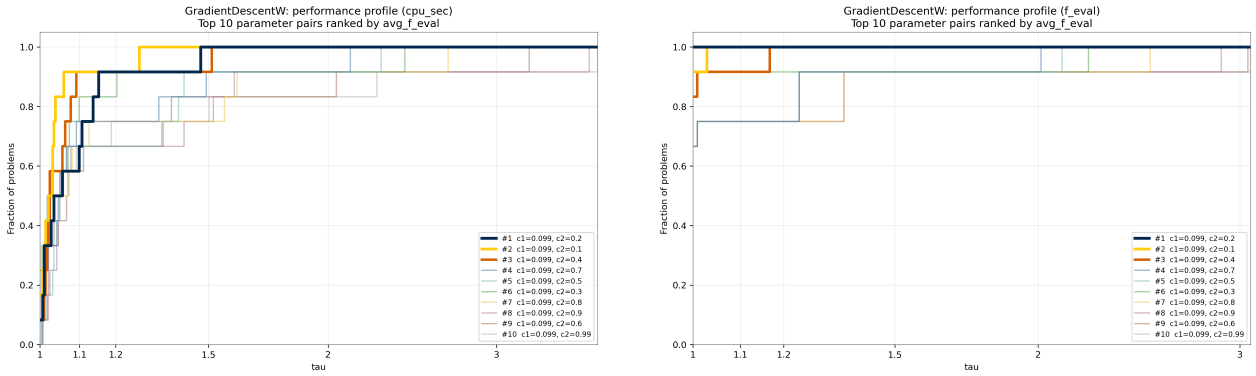


Figure 7: The cpu running time and iterations of Gradient Descent with the top 10 parameter pairs ranked by average performance.

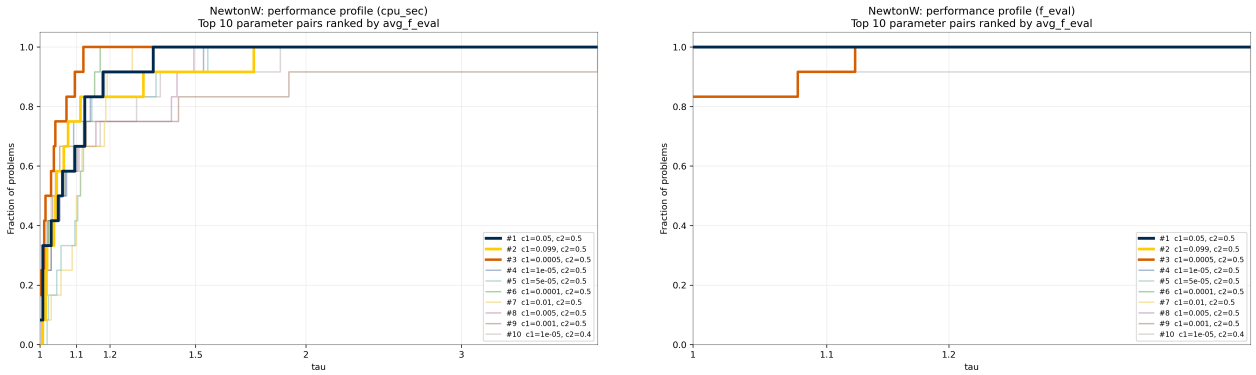


Figure 8: The cpu running time and iterations of Newton method with the top 10 parameter pairs ranked by average performance.

4.4 Sensitivity Analysis

To complement the average-performance and performance-profile study, we further examine the *local* robustness of the selected Wolfe parameter pair for each method. Since c_1 varies across several orders of magnitude whereas c_2 is naturally measured on a linear scale, we first normalize the parameter pair by

$$z_1 = \frac{\log_{10}(c_1) - \log_{10}(c_{1,\min})}{\log_{10}(c_{1,\max}) - \log_{10}(c_{1,\min})}, \qquad z_2 = \frac{c_2 - c_{2,\min}}{c_{2,\max} - c_{2,\min}},$$

and then sample $N = 100$ feasible points uniformly from the disk

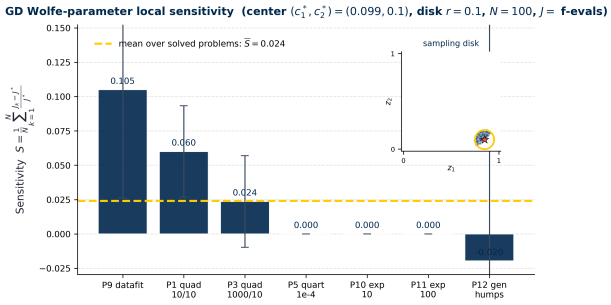
$$\mathcal{D}_r(z^*) = \{z \in [0,1]^2 : \|z - z^*\|_2 \le r\}, \qquad r = 0.1,$$

centered at the current parameter choice $z^* = (z_1^*, z_2^*)$. Any sampled point violating the Wolfe feasibility condition $c_1 < c_2$ is discarded. For each solved problem p, let J_p^* denote the number of function evaluations at the center (c_1^*, c_2^*), and let $J_{p,k}$ denote the value obtained from the kth sampled point. We define the local sensitivity score by

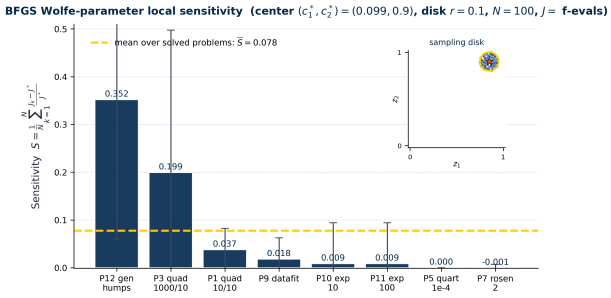
$$S_p = \frac{1}{N} \sum_{k=1}^{N} \frac{J_{p,k} - J_p^*}{J_p^*}.$$

Thus, $S_p \approx 0$ indicates local robustness, a large positive value means that nearby perturbations tend to worsen performance, and a negative value means that some neighboring parameter pairs perform slightly

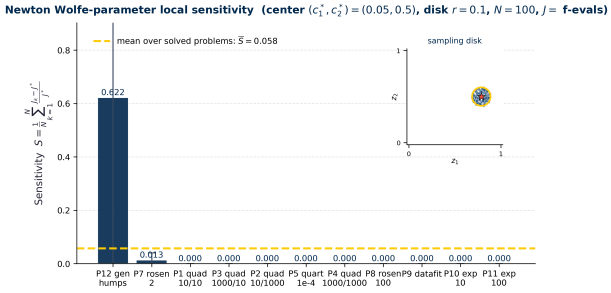
better than the current center. The dashed line in each panel reports the mean sensitivity $\bar{S}$ over the problems for which the center itself converges.



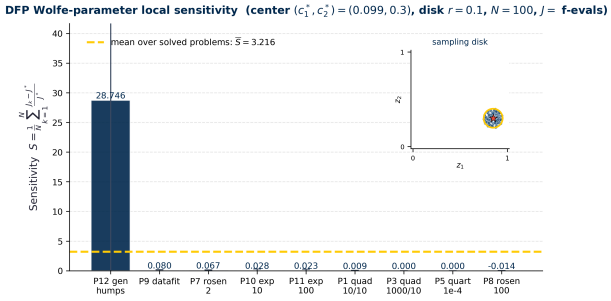
(a) Gradient DescentW



(b) BFGSW



(c) NewtonW



(d) DFPW

Figure 9: Local sensitivity of the selected Wolfe-parameter centers. Bars report the per-problem sensitivity score S_p, and the inset shows the feasible sampling disk in normalized parameter space.

Fig. 9 shows that the four Wolfe-based methods have markedly different local robustness patterns.

Gradient DescentW. Gradient DescentW has the smallest mean sensitivity among the four methods, with $\bar{S} = 0.024$. The largest positive sensitivities occur on P9 datafit (0.105), P1 quad 10/10 (0.060), and P3 quad 1000/10 (0.024), while P5, P10, and P11 are essentially unchanged and P12 is slightly negative (-0.020). This means that, on the problems solved by the center, nearby parameter perturbations usually have only a mild effect. The half-disk shape in the inset is also meaningful. The center $(c_1^*, c_2^*) = (0.099, 0.1)$ lies very close to the Wolfe feasibility boundary $c_1 = c_2$, so a substantial portion of the geometric disk falls into the infeasible region $c_1 \geq c_2$ and must be rejected. Hence the visible "half disk" is caused by the Wolfe constraint rather than by a plotting artifact.

BFGSW. BFGSW exhibits moderate local sensitivity, with $\bar{S} = 0.078$. Most solved problems remain close to zero, but P12 genhumps (0.352) and P3 quad 1000/10 (0.199) stand out clearly, while P1 quad 10/10 (0.037), P9 datafit (0.018), and the exponential problems (0.009 each) show only mild degradation. P7 Rosenbrock is slightly negative (-0.001), indicating that the chosen center is near, but not exactly at, the local best point. Overall, BFGSW is reasonably stable on easy and moderately conditioned problems, but its accumulated curvature information is still noticeably affected by parameter perturbations on ill-conditioned or nonconvex instances.

NewtonW. NewtonW remains highly robust overall, with $\bar{S} = 0.058$. In the current experiment, almost all solved problems have sensitivity exactly zero; the only noticeable deviations are P7 Rosenbrock (0.013) and P12 genhumps (0.622). This means that small perturbations of $(0.05, 0.5)$ leave the number of function evaluations unchanged on nearly the entire benchmark set. The only substantial exception is the nonconvex Genhumps problem, whose geometry makes step acceptance more delicate. Therefore, NewtonW combines

strong average performance with a very flat local sensitivity landscape, which further supports its selection as the algorithm of choice.

DFPW. For the updated center $(c_1^*, c_2^*) = (0.099, 0.3)$, DFPW has mean sensitivity $\bar{S} = 3.216$, still the largest among the four methods. However, this average is overwhelmingly dominated by P12 genhumps (28.746). On the remaining solved problems, the sensitivities are much smaller: P9 datafit (0.080), P7 Rosenbrock (0.067), P10 exponential (0.028), P11 exponential (0.023), and P1 quad 10/10 (0.009), while P3 and P5 are essentially zero and P8 Rosenbrock is slightly negative (-0.014). Therefore, the updated DFP center is not uniformly unstable across all problems; rather, it is reasonably well-behaved on most solved instances but remains extremely fragile on the hardest nonconvex case. This confirms that DFPW is still the least reliable Wolfe-based method from a robustness perspective, even after retuning its parameter center.

Overall, the sensitivity analysis refines the conclusions drawn from the global heatmaps and performance profiles. NewtonW combines strong average performance with the flattest local landscape. BFGSW is moderately sensitive but still acceptable in practice; Gradient DescentW appears stable on the subset it solves, although its best pair lies on the edge of the feasible Wolfe region. And DFPW remains the most fragile method because a difficult nonconvex problem can trigger very large performance degradation. Consequently, if robustness to moderate tuning errors is important, NewtonW remains the strongest practical choice among the four Wolfe line-search methods.

5 Conclusion

This report presented a systematic computational comparison of ten classical continuous optimization algorithms on twelve benchmark problems with different dimensions, conditioning levels, and degrees of nonconvexity. Overall, the results show a clear hierarchy: methods using second-order information, especially Newton-type and trust-region methods, usually achieve the fastest convergence and the best final accuracy, while first-order methods are much more sensitive to ill-conditioning and curved valleys. Quasi-Newton methods provide a useful middle ground, and among them BFGS is consistently more reliable than DFP.

We also investigated the Big Question of Wolfe-parameter selection for the four Wolfe-based algorithms. The global heatmaps and performance profiles show that there is no universally optimal pair (c_1, c_2) for every method and every problem. instead, the preferred choice depends strongly on the algorithm and should be evaluated together with local robustness. Our final parameter recommendations are $(0.099, 0.1)$ for Gradient DescentW, $(0.05, 0.5)$ for NewtonW, $(0.099, 0.9)$ for BFGSW, and $(0.099, 0.3)$ for DFPW. The local sensitivity study adds an important refinement. NewtonW is the most locally robust around its selected center, BFGSW shows moderate sensitivity, Gradient DescentW is locally stable on the problems it solves but lies close to the feasibility boundary $c_1 = c_2$, and DFPW remains the most fragile because its performance can deteriorate sharply on difficult nonconvex problems.

From both the performance and implementation perspectives, we would declare NewtonW the overall winner of this project. It gave the best balance of speed, robustness, and solution quality on our benchmark set. However, it was not the easiest method to code, since Newton-type methods require Hessian information and additional safeguards. Gradient descent was the easiest algorithm to implement and tune, but its performance was clearly weaker. BFGS offered the best compromise between coding difficulty and optimization performance, while DFP was generally less stable in both behavior and final results.

Our recommendations therefore depend on the user. For an expert coder who can compute second derivatives reliably, NewtonW is the strongest choice. For a user who is not an expert in coding or nonlinear optimization, BFGS is the more practical recommendation, since it avoids explicit Hessians while still performing well. If second derivatives are unavailable, BFGS becomes the preferred method, while gradient descent remains the simplest baseline when ease of implementation is the main concern. Overall, this project shows that a good optimization method should be judged not only by raw performance, but also by robustness, tuning burden, and implementation cost.