Project Assignment 1, Unconstrained optimization methods

Tuomas Porkamaa, Anton Saukkonen
Aalto University School of Science, Department of Mathematics and Systems Analysis
Aalto University School of Science, Department of Mathematics and Systems Analysis
{*tuomas.porkamaa, anton.saukkonen*}*@aalto.fi*

1 Project description

1.1 Aim

The aim of this project is to implement and test unconstrained optimization methods that have been presented and taught in the Nonlinear Optimization course. Namely, we consider the following functions:

$$f_1(x_1, x_2) = 2(0.5x_1^2 + 4x_2^2) - 0.5x_1x_2 \tag{1}$$

$$f_2(x_1, x_2) = e^{x_1+2x_2-0.1} + e^{x_1-2x_2-0.1} + e^{-x_1-0.2} \tag{2}$$

$$f_3(x_1, x_2) = (x_1^2 + x_2 - 10)^2 + (x_1 + x_2^2 - 15)^2 \tag{3}$$

$$f_4(x) = \frac{1}{2}x^T Ax - b^T x, x \in \mathbb{R}^{100} \tag{4}$$

and several unconstrained optimization algorithms in order to study their behaviour via analysis of convergence in terms of optimality, efficiency and conditions imposed by topological properties of the function in question. In particular, the project is split into four subtasks:

1. For the function (1) we start from the point $(x_1, x_2) = (-10, 10)$ and apply two unconstrained optimization methods: *Gradient method* and *Gradient method with momentum* (aka 'Heavy ball') in order to find the optimal solution. In addition to that, each of the aforementioned methods is implemented in two versions, namely, using *exact* and *inexact* line search methods for the choice of optimal step size. The line search algorithms are *Bisection search* and *Armijo rule*.
2. For the function (2) we start from the point $(x_1, x_2) = (-2.5, -3.5)$ and apply another two unconstrained optimization methods: *Newton method* and *Broyden-Fletcher-Goldfarb-Shanno method* in order to find the optimal solution. Analogously, each of the methods is implemented using *exact* and *inexact* line search algorithms for the choice of optimal step size, respectively *Bisection search* and *Armijo rule*.
3. For the function (3), we start from the point $(x_1, x_2) = (-0.5, -2)$ and apply *Gradient method*, *Gradient method with momentum*, *Newton method* and *Broyden-Fletcher-Goldfarb-Shanno method* in order to find the optimal solution. However, in this case each of the algorithms were utilizing only the exact line search method for the choice of optimal step size, namely *Bisection search*.
4. For the function (4), 100 random instances of positive-definite A and b were generated. Furthermore, *Gradient method*, *Gradient method with momentum*, *Newton method* and *Broyden-Fletcher-Goldfarb-Shanno method* were employed for each of the randomly generated problems, while each of the algorithms utilized exact and inexact line search methods, *Bisection search* and *Armijo rule*

respectively. Additional experimental aim in the subtask 4 was to observe how condition numbers of matrix A influences the behaviour of the algorithms.

1.2 Technical set up

Concerning the technical details related to the computational hardware and software, two alternatives were considered. Namely, standard desktop computer and Aalto CS Jupyterlab server with Julia kernel 1.6.2. The latter was chosen due to efficiency and standardization reasons.

1.3 Means of benchmark

The computational times were obtained via utilization of the in-build Julia macro **@benchmark** from the package **BenchmarkTools**. In sections 2.1-2.3, the realization of algorithm on the particular function is executed 10000 times with 1 evaluation, and after that, average results are reported. In the analysis, we have only considered average measure, disregarding standard deviation, minimim, maximum and median. Mean time per iteration were calculated via averaging mean time reported by **@benchmark** with number of iterations it takes to converge to the extrema point. Since the functions are deterministic and number of iterations to converge was known before benchmarking, the approach gives reliable results in terms of average values.

1.4 Special details related to implementation and test algorithms

The optimization algorithms as well as line search methods have been implemented according to the instructions provided in project statement. Thus, parameters for Armijo rule, Bisection search, Gradient method and Newton method have been realized as they were provided. The rationale for choosing weight parameter in Heavy ball is described in the section 2.1.

2 Numerical results

2.1 Sub-task 1

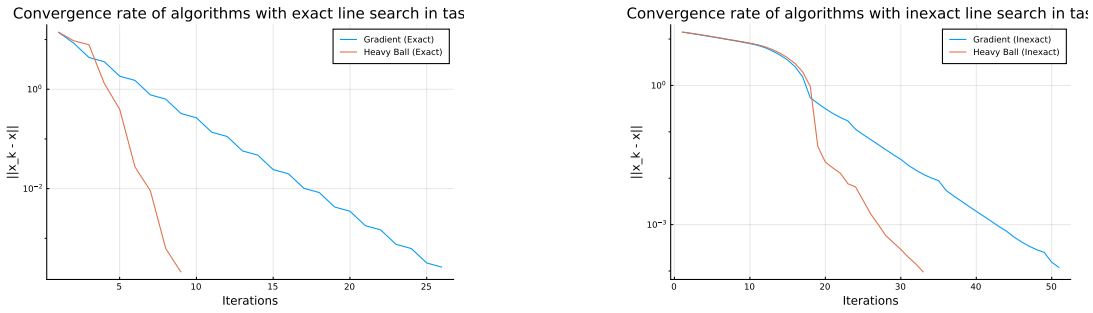
We start our analysis by discussing the behaviour of Gradient method with momentum. First of all, the necessity to employ parametrized Gradient method is to eliminate effect of zig-zagging observed in the standard Gradient method. There is no common opinion and algorithm that is scientifically acknowledged to always produce robust zig-zagging reduction. However, it was observed that there is connection between the zig-zagging rate and the optimal value for the parameter w, which naturally depends on the function itself and also the line search method employed. In the current project, it was decided that the most effective way to choose parameter w is to fine-tune it manually. Namely, around 10 different ranges for the potential values of w have been tested. For the function in question, some boundary cases of the results are summarized in the Table 1, where it is seen that certain values of the weight parameter result in significantly better convergence speed compared to the gradient method without momentum, while some of them, on the other hand, drastically worsen performance.

Table 1: Convergence results of algorithms in part 1

Method	Value of w	Time(T/I)	Iterations	Solution	Argument
Heavy-ball (Bisection)	0.1	745/25 μs	30	4.11e-8	[-0.00019, -2.35e-5]
Heavy-ball (Bisection)	0.3	228/28 μs	8	4.35e-8	[-0.0002, 3.90e-6]
Heavy-ball (Bisection)	0.5	446/28 μs	16	5.53e-8	[-0.0002, 3.39e-5]
Heavy-ball (Bisection)	0.9	2038/32 μs	62	9.89e-8	[-0.0003, 2.13e-5]
Heavy-ball (Armijo)	0.1	387/9 μs	44	2.28e-8	[9.93e-5, 4.34e-5]
Heavy-ball (Armijo)	0.19	262/8 μs	32	2.43e-8	[8.30e-5, -4.42e-5]
Heavy-ball (Armijo)	0.5	287/8 μs	37	3.09e-8	[-1.02e-5, -6.23e-5]
Heavy-ball (Armijo)	0.9	861/11 μs	77	1.60e-7	[0, -3.04e-6]
Gradient (Bisection)	*	632/25 μs	25	6.93e-8	[-0.0002, 3.56e-6]
Gradient (Armijo)	*	478/9.5 μs	50	3.32e-8	[-0.0001, -5.64e-5]

In particular, it could be seen that the most optimal weight values for the Heavy Ball with exact and inexact line search methods are 0.3 and 1.9 respectively. Observe that Heavy ball with exact line search outperforms Gradient method with exact line search almost by the factor of 3, which accumulates to 8 vs. 25 steps respectively. Similarly, for the algorithms with inexact line search methods, the heavy ball is faster roughly by the factor of 2 displaying 32 vs 50 iterations. On the other hand, whenever values of w were chosen close to 1, both algorithms took more steps to converge (62 and 77), which resulted in significantly longer computational times.

Another important observation could be deduced from results summarized in the Table 1 and support the theory. Namely, it is observable that algorithms, which employed inexact line search method, namely, Armijo rule, yielded faster convergence, however, eventually took more steps to arrive to the optimal point. This can be explained by the fact that Armijo rule is more efficient due to less complicated optimization procedure, however, the optimal step size is sacrificed, which results in higher amount of steps taken to converge. This claim can be also supported by results summarized in Table 1. Namely, average computational time during one iteration with Armijo rule does not exceed $11\mu s$, while with exact line search method, average computational time per iteration may reach up to $32\mu s$. Convergence rates for algorithms with exact and inexact line search method presented on the Figure 1. Observe again the advantage in performance of Heavy ball due to elimination of zig-zagging.



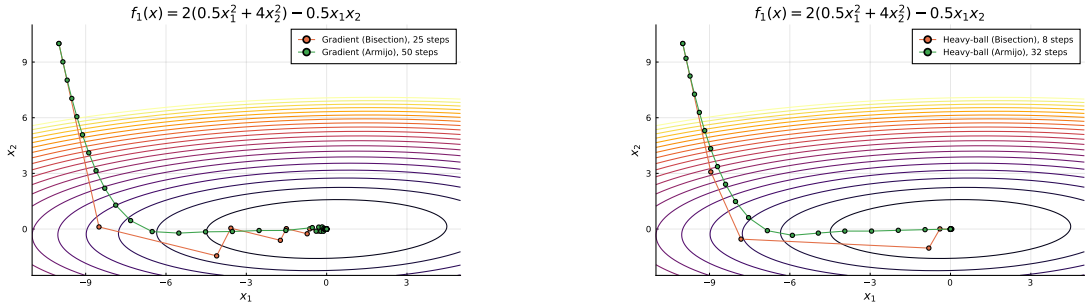
(a) **(b)**

Figure 1: Convergence rates for algorithms on f_1 with exact/inexact line search

Next we will study the topological nature of f_1 in order to reason about optimality and further complement convergence rate arguments. Firstly, observe that

$$f_1(x_1, x_2) = 2(0.5x_1^2 + 4x_2^2) - 0.5x_1x_2 = 0.5x^T Hx, \quad H = \begin{bmatrix} 2 & -0.5 \\ -0.5 & 16 \end{bmatrix}$$

Thus, Hessian of f_1 is matrix H. By computing its eigendecomposition, we find out that eigenvalues $\lambda_{max} = 16.01, \lambda_{min} = 1.98$ are positive, which is implies that H is positive definite and thus f_1 is a convex function. This implies that whenever local optimum is obtained, it is also a global optimum point. Observe that this is the case by checking gradient of f_1 and applying first order optimality conditions. Hence, it implies that all algorithms converged to the global optimum. Convexity could also be infered from the nature of the level curves presented in the Figure 2.



(a) Gradient method exact/inexact

(b) Heavy Ball ($w = 0.3/0.19$) exact/inexact

Figure 2: Trajectory plot of f_1

Furthermore, since f_1 is quadratic function of the form $\frac{1}{2}x^T Hx$ with PD matrix H, by the Theorem 6 from lecture notes, it follows that Gradient method with exact line search converges linearly and particularly, rate of convergence is bounded by

$$\frac{f(x_{k+1})}{f(x_k)} \leq \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}\right)^2 = 0.61$$

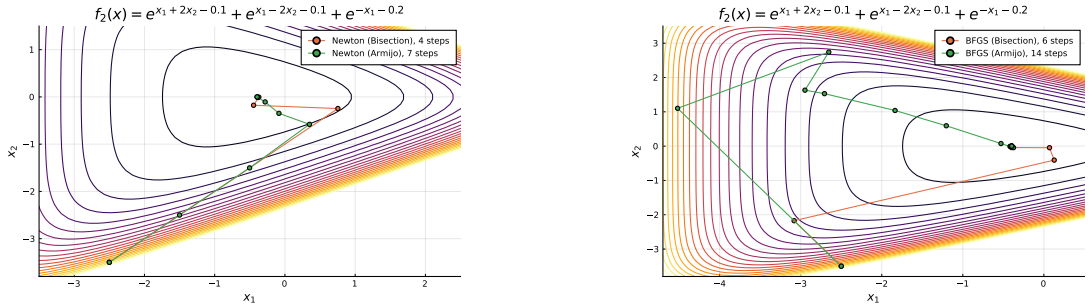
for every k. This shows that convergence rate of Gradient method is moderate. Finally, by examining condition number of H, we found out that $\kappa(H) = \frac{\lambda_{max}}{\lambda_{min}} = 8.08$, which implies that H is relatively well conditioned and the algorithms should be numerically stable, which is indeed observable from the experiments.

2.2 Sub-task 2

Table 2: Convergence results of the algorithms in part 2.

Method	Time(T/I)	Iterations	Solution	Argument
Newton (Bisection)	130/33 μs	4	2.434	[-0.3966, -0.0001]
Newton (Armijo)	92/13 μs	7	2.434	[-0.3964, -0.0001]
BFGS (Bisection)	221/37 μs	6	2.434	[-0.3966, -0.0001]
BFGS (Armijo)	176/13 μs	14	2.434	[-0.3964, 0.0]

To start off with topological properties of f_2, we can see that the function is a sum of components formed as $g(h(x))$ where due to convexity of h and g, the convexity is also preserved for the sum of their compositions. Therefore, the point $\bar{x}$ satisfying $\nabla f_2(\bar{x}) = 0$ is indeed the global optimum, which in this case equals to $\bar{x} = (-\frac{0.1+log(2)}{2}, 0) \approx (-0.3966, 0)$. The convexity also gives a reason to utilize second-order approximation based optimization algorithms because without this property, the local approximation could lead us to saddle point or maximum. As we can see from table 2, each method successfully converged near the optimal solution so the further comparison will focus on the efficiency of the algorithms in terms of iteration count and average time spent to find the optimal solution.



(a) Newton's method, exact/inexact

(b) BFGS, exact/inexact

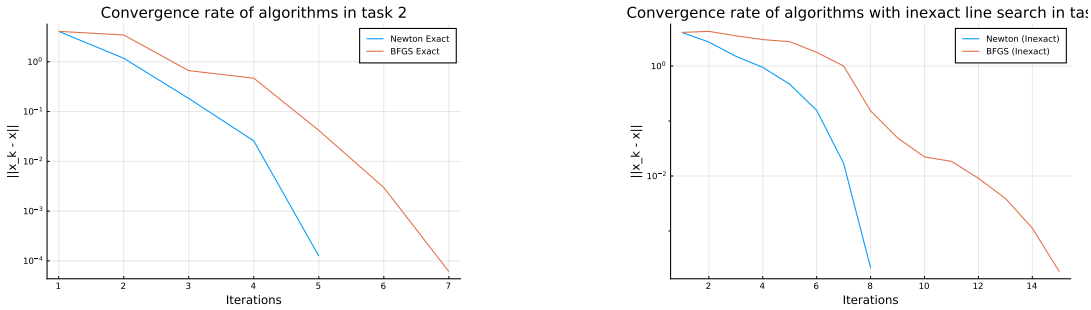
Figure 3: Convergence trajectory plots for f_2.

The first implemented method was Newton's method utilizing Bisection and Armijo line searches. From table 2 we can observe that the inexact Newton's method vanquished its exact counterpart significantly in terms of average time, despite that it required more steps to converge. This is of course expected for inexact line searches because they are designed to be relatively time-efficient at the expense of finding the accurate optimum. The behaviour of inexact Newton can be seen in Figure 3 a, where the first two steps were clearly underestimated. The trajectory of the exact counterpart shows us that in the first direction, the line function minimum is quite far from the starting point, the distance being much larger than the length of the direction vector scaled by the maximum step size $\lambda = 1$ used in inexact line search implementation. Tuning parameter λ would remove this stepping behaviour, but it would most likely cause harmful effects when the algorithm proceeds closer to optimal solution.

The other implemented method in this section was quasi-Newton variant called BFGS that use iteratively updated symmetric and positive-definite approximation of the Hessian of function. This iterative approach makes BFGS quite efficient especially in high-dimensional problems because the method utilizes previously calculated quantities. However, by looking at table 2 we see that there is no notable differences of per-iteration computation times between methods of corresponding line searches. This might happen because the quadratic complexity of the Hessian matrix calculation in low dimensions is not that expensive. Within the scope of total computation time, the Newton's method still outdoes the BFGS because of the lower number of iterations needed for convergence. The iterative improvement of the Hessian approximation becomes visible when looking the first steps in Figure 3 b, where the search direction stabilizes after first three steps for both implemented line search methods. After these first iterations, it can be assumed from the trajectory plots that

the Hessian approximation is already reasonably close to the ground truth.

The more detailed iteration specific convergence for the variants of both methods is visualized in Figure 4. In plot a, we can see the the typical quadratic and superlinear convergence rates are achieved using exact line search in Newton's method and BFGS respectively. When using inexact line search, in plot b, the Newtons method still continues showing quadratic convergence rate, but for first 5 iterations, the convergence of BFGS is quite far from superlinear rate. This behaviour was explained in previous paragraph, but the convergence rate plot shows it more explicitly.



(a) **(b)**

Figure 4: Convergence rates for algorithms on f_2 with exact/inexact line search.

2.3 Sub-task 3

We start our analysis by discussing topological properties of the f_3. First of all, observe that function is not convex. To support the claim, we show that property of convexity is not preserved since the function has 4 minima points. We can compute them directly:

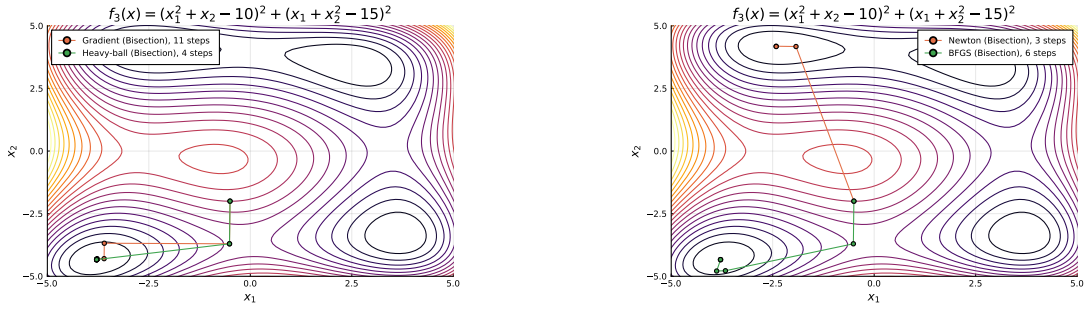
$$(x_1, y_1) \approx (-3.78, -4.33)\,; (x_2, y_2) \approx (3.65, -3.36)\,;$$
$$(x_3, y_3) \approx (2.54, 3.52)\,; (x_4, y_4) \approx (-2.41, 4.17)\,;$$

After evaluating the function at $(x_i, y_i)\,, i = 1, ..., 4$ we observe that $f(x_i, y_i) = 0$ for every i. Observe that for every $x \in \mathbb{R}^2$

$$0 \leq f_3(x)$$

Thus the function is bounded below by zero and since there exists $x \in \mathbb{R}^2$, such that $f_3(x) = 0$ we conclude that there is no unique global minima and $(x_i, y_i).i = 1, ..., 4$ are minimums of f_3. The existence of multiple extrema points implies that convergence of particular algorithm depends on the choice of initial point. From the results summarized in the Table 3, it is observable that Heavy ball with all choices of parameter w, Gradient Method and BFGS have converged to the point (x_1, y_1), however, Newton method converged to the point (x_4, y_4). One can verify that the choice of the initial point is the crucial factor, by investigating level curve plots presented in the Figure 5 and selecting the initial point to be close to the one of critical or saddle points. For instance, by selecting the initial point for Newton's method to be $(-3, -3)$, the method converges to the point

(x_1, y_1). One possible explanation for this behaviour could be the fact that interval for step size λ in this case is relatively narrow, namely $[0, 10]$. For the sake of experiment, we can expand the interval to $[0, 1000]$ and observe that Newton method converges to (x_2, y_2), which is still different from the result obtained by another methods. The reason for this behaviour is lying the nature of Newton method. Namely, minimum of quadratic approximation, which is used during the computation of Newton method might be significantly different from the actual minimum of a function and in this case, effect is amplified by the fact that global minima is not unique.



(a) Gradient method and HB ($w = 0.2$) exact

(b) Newton method and BFGS exact

Figure 5: Trajectory plot of f_3

Next, we discuss the performance of the algorithms. Firstly, note that the same principle of empirically choosing weight parameter for Heavy Ball was employed here as well and in general, we could observe the same phenomena also in this problem. Namely, as seen in the Table 3, carefully chosen value of weight parameter could either significantly improve the performance or on the other hand worsen it a lot. For instance, it is observable that Heavy Ball method with carefully fine-tuned parameter $w = 0.2$ outperforms all other algorithms resulting only in 4 steps and $90\mu s$ of total computational time required to converge. This is significantly faster than Gradient method without momentum as well as BFGS. Observe that although Gradient method and Heavy ball posses linear convergence rate, while BFGS is known to have superlinear theoretical convergence, elimination of zig-zagging effect allows to outpeform theoretically faster algorithm. The claim can be supported by the Figure 5 (a), where one can observe the absence of zig-zagging in Heavy-Ball case. Furthermore, as the results of Table 3 suggest, that quadratic convergence rate of Newton method is indeed seen in the experiments. Moreover, as seen in the Figure 6, in terms of iteration's count, it manages to outperform Heavy ball (which yields the minimum total computational time) and all other methods. However, the overhead of calculating inverse of Hessian in Newton method is clearly seen in the average computational time per iteration, which is $34\mu s$ and in this case turns out to be the longest one (see Table 3). Finally, in general we note that while Newton and BFGS methods take less steps to converge, average computational time per iteration is fairly longer. This happens due to the fact that methods obtain inverse of the Hessian either by computing it directly or by approximating it. The overall convergence rate comparison in terms of iteration count is presented in the figure Figure 6.

Table 3: Convergence results of the algorithms in part 3.

Method	Value of w	Time(T/I)	Iterations	Solution	Argument
Heavy-ball (Bisection)	0.1	162/20 μs	8	3.04e-10	[-3.78, -4.33]
Heavy-ball (Bisection)	0.2	90/22 μs	4	1.38e-11	[-3.78, -4.33]
Heavy-ball (Bisection)	0.5	352/19.5 μs	18	6.87e-10	[-3.78, -4.33]
Heavy-ball (Bisection)	0.9	2046/19.5 μs	105	5.09e-9	[-3.78, -4.33]
Gradient (Bisection)	*	229/21 μs	11	3.25e-10	[-3.78, -4.33]
Newton (Bisection)	*	103/34 μs	3	2.62e-11	[-2.41, 4.17]
BFGS (Bisection)	*	198/33 μs	6	5.62e-16	[-3.78, -4.33]

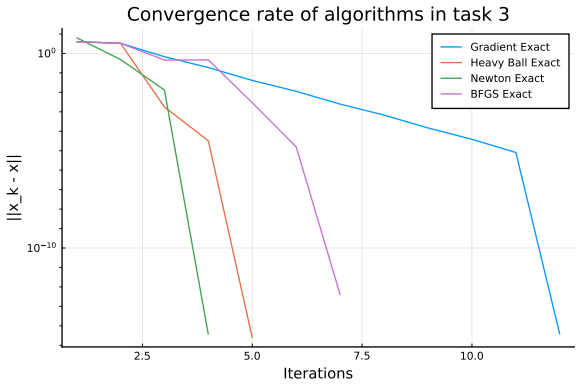


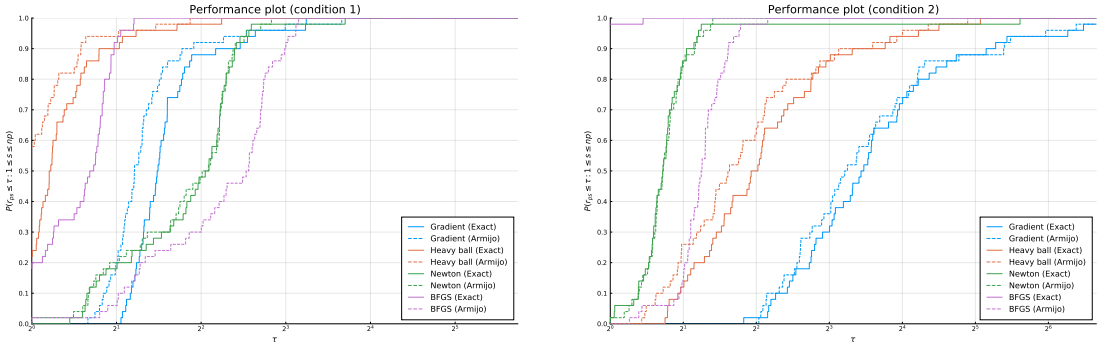
Figure 6: Convergence rates for algorithms on f_3

2.4 Sub-task 4.

In the fourth task we compared all previously introduced methods on a significantly larger set of quadratic problems defined as (4). The problems differ by the parameters A and b that in each case were randomly generated, and by the condition number of A. This totals to set of 100 problems P to solve. Furthermore, the entire problem set was divided to two individual sets $P = P_1 \cup P_2$ where $|P_i| = 50$ and in P_1 smaller condition numbers were used for A. The computing time was used as a performance metric.

The figure 7 a shows the performance profiles for all solvers for problem set P_1. By inspecting the curves at $\tau = 1$ we can observe that the highest probability, with a significant gap to the others, was obtained by inexact Heavy ball solver, where $\rho_{HB_A}(1) \approx 0.58$. This indicates that approximately 58% of the problems in P_1 were solved fastest using inexact Heavy ball. The second largest winning probabilities were obtained by exact Heavy ball and BFGS, and for the rest of the solvers, the winning probabilities were ≈ 0. We can also see that for any $\tau \geq 1$ Heavy ball and exact BFGS solve more problems within a fraction of τ of any other solver time. Especially when $\tau \in [1, 2^{0.5}]$ these methods were almost without exception the only ones being within τ factor of fastest solving time. The gradient descent methods also converged moderately fast compared to Newton's method on many problems on this set for $\tau \in [2, 2^{2.4}]$. The overall good performance of gradient descent based methods is the result of small condition numbers of A. Because of the low condition numbers, the level curves of f_4 are more circular indicating that the negative gradient points more accurately towards the global minimum and thus allows faster convergence.

The figure 7 b shows the performance profiles for all solvers in problem set P_2. Here in terms of winning probability we can already see that the exact BFGS method performs overwhelmingly well compared to others, with the probability $\rho_{BFGS_B}(1) \approx 0.98$. In addition, just after tiny increase in τ, the performance profile probability converges to 1, indicating that the computing time of exact BFGS is always within the factor of $\approx 2^{0.5}$ of the fastest time for every problem in P_2. The other noticeable feature is that the second-order approximation based methods solve the problems relatively faster than the gradient based methods which is almost opposite behaviour compared to problems in P_1. The unfitness of gradient based methods shows also the convergence of the probability of traditional gradient solver $\rho_G(\tau)$. Even for $\tau \geq 2^6$ there still exist some problems whose solving time is greater than fastest time multiplied by a factor of τ. The relatively poor performance of gradient descent methods can be argued using opposite argument that was used for problems in P_1. Because now the condition number is larger, the level curves are more stretched and the negative gradient can actually point to direction that is not well aligned with the direction where the global minimum is located.



(a) Smaller condition numbers

(b) Larger condition numbers

Figure 7: Performance profiles for all methods

3 Discussion and conclusions

In **Task 1**, the function was convex, and thus, all algorithms have converged to global optimum. With respect to the theory, all methods possess linear theoretical convergence rate. However, it was observed that due to fact that Armijo rule sacrifices optimality of step size in favor of overall efficiency, algorithms employing it generally converge faster, even though make more steps. This is explained by the fact that average time per iteration is generally lower for inexact than for exact line search methods or at least as low. However, there is a tradeoff between number of steps taken and total computational time. Finally, we observed that Heavy ball with correctly fine-tuned parameter performs really well with both line search methods by elimination of zig-zagging effect. On the countrary, inaccurately chosen value w may drastically damage performance. Average time per iteration for all methods is approximately in the same ballpark, but depends on whether the exact or inexact line search is used.

In **Task 2**, the function was convex and therefore there should not be any prominent convergence issues with Newton's method and BFGS when line search functionality is used. In BFGS, the

identity matrix was used as an initial approximate of the Hessian but due to the iterative matrix updates, the approximation converged reasonably close to the true Hessian after few steps. The effect of poor direction choices at the beginning were amplified by the inexact line search that even caused the algorithm to move further away from the global optimum before correcting its direction. The measured per iteration times between methods were approximately equivalent because the effect of exact Hessian calculation becomes noticeable on more higher dimensions than this problem was defined. However, because BFGS required a larger number of iterations to convergence, the difference of times become noticeable in favor of Newton's method.

In **Task 3**, the function was not convex, but exhibited ellipticity properties and was bounded by zero from below with 4 non-unique minima points. Thus, the choice of initial point was the crucial factor in the convergence of the algorithms and it was observed that convergence points were different for some of them. Performance wise, it was observed that carefully fine-tuned weight parameter of Heavy ball with theoretical linear convergence rate is able to outperform methods with quadratic convergence rate, which partially depends on the topological nature of the function in question and choice of initial point. Finally, we saw that average time per iteration for second and quasi-second order methods is higher due to more expensive computations related to Hessian matrix. Similarly with Task 1, the average time per iteration for all methods is in the same range for method of the same order.

In **Task 4**, we carry out more comprehensive analysis of the convergence properties by applying all 8 variants of the 4 methods to a larger set of quadratic problems in $\mathbb{R}^{100}$. The problems were divided to two sets of size 50 that differ by the condition numbers of the matrices A in the function. For both of the problem sets, we evaluated the methods by using performance profiles. For small-conditioned problems, we observed that the inexact Heavy ball method was fastest method for approximately 58% of problems, following by the exact Heavy ball and BFGS by corresponding proportions of $\sim 21\%$ and $\sim 20\%$. Overall, these three methods were on average the best performing ones in this problem set by all criteria, but the Gradient descent also showed moderate convergence speed. When the methods were applied to the large-conditioned problems, we obtained different results. The gradient based approaches were all outperformed by Newton's method and BFGS. Especially the exact BFGS showed remarkable performance in terms of winning probability of 0.98 and the convergence times of this method were always bounded by a factor of $2^{0.5}$ of fastest convergence speed on all problems. The computation times of Gradient descent had the lowest probability of being within the fastest by almost all factors. All of these results can be expected from the theoretical properties of small and large conditioned quadratic problems.