Mat II – Gradient Descend II Report

Sebastijan Trojer

April 14, 2025

1 Introduction

In this homework we were tasked with implementing gradient descend (GD), Polyak GD, Nesterov GD, AdaGrad GD, and SGD along with Newton, BFGS and L-BFGS. We evaluated and compared the first four on 3 different convex functions and the last four on a simple linear regression task.

2 Methodology and results

2.1 Part I - GD, Polyak GD, Nesterov GD and AdaGrad

We evaluated the aforementioned GDs on the following functions and starting points:

1. $f(x, y, z) = (x - z)^2 + (2y + z)^2 + (4x - 2y + z)^2 + x + y$
   Starting points: $(0, 0, 0)$ and $(1, 1, 0)$.
2. $g(x, y, z) = (x - 1)^2 + (y - 1)^2 + 100(y - x^2)^2 + 100(z - y^2)^2$
   Starting points: $(1.2, 1.2, 1.2)$ and $(-1, 1.2, 1.2)$.
3. $h(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$
   Starting points: $(1, 1)$ and $(4.5, 4.5)$.

For each function and each GD variations we performed 2, 5, 10 and 100 steps and compared the convergences. The results are given in Table 1. The true minimums for the functions are the following: $f : -0.19541@(-0.1515, -0.2121, 0.1515)$, $g : 0.0@(1, 1, 1)$ and $h : 5.2124 \times 10^{-5}@(3.0, 0.5015)$.

The functions we tested the methods on are very steep, so in some cases the learning rate had to be set to a really low value, to prevent overshooting. The overall best performing approach was Nesterov, however the Polyak was really close. Base GD also performed good overall, however especially on the really steep functions (h) it converged significantly slower than the other two. The by far worst performance was achieved by AdaGrad, as it often seemed to get stuck due to the accumulating weights and only performed good on the first starting point of f. Even better performance could be achieved for Polyak and Nesterov

Method	Steps	$f(x,y,z)$		$g(x,y,z)$		$h(x,y)$	
		SP1	SP2	SP1	SP2	SP1	SP2
GD	2	-0.0351	6.0553	8.1063	8.8489	14.0487	766.5966
Polyak GD	2	-0.0359	5.8735	7.9409	8.6296	14.0409	317.3669
Nesterov GD	2	-0.0358	5.9158	7.9561	8.6664	14.0409	337.8935
AdaGrad	2	-0.0315	10.5644	11.5607	13.7525	14.1984	174808.13
GD	5	-0.0715	3.7295	4.8997	5.8598	13.8183	573.3181
Polyak GD	5	-0.0739	3.4667	4.5013	5.5520	13.7842	234.1842
Nesterov GD	5	-0.0737	3.4908	4.5383	5.6033	13.7842	251.2675
AdaGrad	5	-0.0555	10.1975	11.5266	13.7113	14.1943	174803.57
GD	10	-0.1098	1.9655	2.2744	4.5401	13.4382	391.0198
Polyak GD	10	-0.1129	1.7118	1.9040	4.4221	13.3593	180.1409
Nesterov GD	10	-0.1128	1.7301	1.9354	4.4393	13.3592	191.6422
AdaGrad	10	-0.0792	9.7964	11.4883	13.6650	14.1896	174798.44
GD	100	-0.1945	-0.1466	0.0192	4.2104	8.1100	41.4483
Polyak GD	100	-0.1951	-0.1643	0.0186	4.2098	7.3269	27.3436
Nesterov GD	100	-0.1951	-0.1640	0.0186	4.2099	7.3240	28.1309
AdaGrad	100	-0.1596	7.8908	11.2888	13.4244	14.1653	174771.48

Table 1: Results of different methods on the functions. Each function column is split by the two different starting points: (0,0,0), (1.2,1.2,1.2), and (-1,1.2,1.2) for f, g, and h, respectively. Learning rate and μ were fixed at $0.01 \cdot 0.99^k$ and 0.1. For g and h the base learning rate was reduced by a factor of 100. For the second starting point of h base learning rate was set to 0.00001.

by finetuning the μ value, however the results still show the trend. AdaGrad could be improved, but it would require additional learning rate tuning.

Furthermore, we compared the time efficiency of the methods. We compared the performances given 0.1, 1 and 2 seconds. Again, Polyak and Nesterov outperformed both GD and AdaGrad. For f all methods except AdaGrad converged close to the minimum in less than 100 miliseconds for both points, with AdaGrad getting stuck at function value 0. For g, all the methods converged to a value in less than 0.1 second, so the results were again the same for all the time limits for both starting points. The same occured for h, as even in 0.1 seconds, the Polyak GD performed 10300 steps, which is enough for convergence, especially with a decaying learning rate. If we picked a larger decay or a different learning rate method, we could probably make the convergence even closer. Overall the best performing were Polyak and Nesterov GDs.

2.2 Linear regression

In this part we implemented linear regression and fitted a line to the data using learning rate of 0.001×0.99^k. We tried to fit both k and intercept, so 2 parameters. We initialized k to -1 and intercept to -1. All methods converged to the neighborhood of $k = 1$ and intercept between 0 and 1, regardless of N.

The main effect that sample size had was outlining the performance and how much the final loss was minimized. We performed 1000 steps of descent on N = 1 000 000 and the following happened: BFGS and L-BFGS converged and took 19 seconds, GD, Newton and SGD tried to fix the intercept rather than k so it didn't fit the line well, and SGD failed to fit any parameters. We tested all approaches on multiple N and learning rates and the only ones that worked consistently were the BFGS and L-BFGS, where both achieved a very small loss.

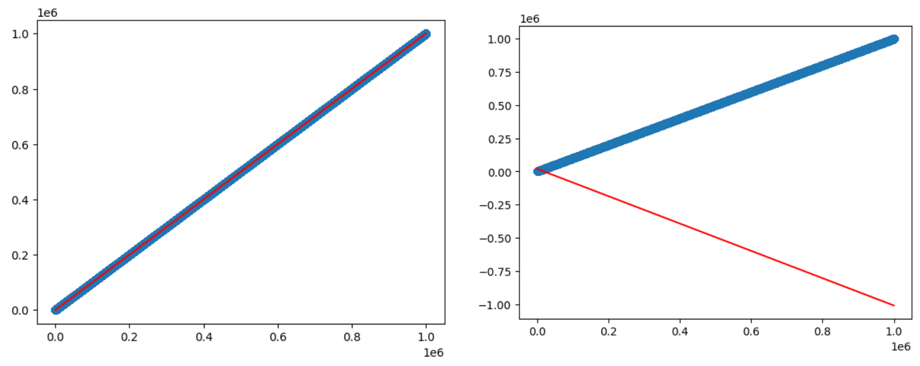

Figure 1: Visualization of the fitted line on N=1 000 000 with BFGS (left) and Newton (right).

3 Conclusion

We implemented 8 different gradient descent methods and tested them in different scenarios and analyzed their failure points. Overall the most robust for steep convex functions are Polyak and Nesterov, while overall best algorithm is the BFGS, which does a rank-1 approximation of the Newton.