

Numerical Optimization - Handin 5

This report seeks to evaluate our implementation of the steepest descent algorithm and Newton's algorithm both modified to optimize under linear equality constraints.

1 Theory

Here we derive the stopping criterion we use in our implementation along with a proof of an important lemma used to modify the algorithms for linear equality constraints.

1.1 Finding an Analytic Stopping Criterion

$$\begin{aligned}
 n(\lambda) &\stackrel{\text{def}}{=} \|\nabla f(x_k) - A^\top \lambda\|^2 \\
 \implies \nabla n(\lambda) &= -2A(\nabla f(x_k) - A^\top \lambda) = \mathbf{0} \\
 \iff A\nabla f(x_k) &= AA^\top \lambda \\
 \iff (AA^\top)^{-1}A\nabla f(x_k) &= \lambda \stackrel{*}{=} \underset{\lambda}{\operatorname{argmin}} n(\lambda) \quad (\star) \text{ } n(\lambda) \text{ is convex}
 \end{aligned}$$

The inverse of AA^\top exists because A has full row-rank.

1.2 Proving Lemma 7

The lemma states that given an n -dimensional optimisation problem with m linear equality constraints, $c_i(x) = a_i^\top x + b_i$, $i = 1, \dots, m$ with $\|a_i\| > 0$ and no inequality constraints. Then the tangent cone is given by:

$$T_C(x) = \{p \in \mathbb{R}^n \mid \forall i : a_i^\top p = 0\}$$

Proof:

Firstly we show $T_C(x) \subseteq \{p \in \mathbb{R}^n \mid \forall i : a_i^\top p = 0\}$. Consider an arbitrary $p \in T_C(x)$. Per definition of $T_C(x)$ this means there exists a $z(\alpha) : [0, u] \mapsto \mathcal{C}$ with $z(0) = x$ and $z'(0) = p$. Since z maps to the feasible set \mathcal{C} we know that the z -values satisfies the constraints:

$$\forall i : a_i^\top z(\alpha) + b_i = 0 \stackrel{(\alpha=0)}{\implies} \forall i : a_i^\top z'(0) = a_i^\top p = 0$$

The implication above follows from differentiating both sides but we constrain the expression on the right hand side to $\alpha = 0$ since this is the only place we are guaranteed that z is differentiable. Since the the above holds for any $p \in T_C(x)$ we have shown that $T_C(x) \subseteq \{p \in \mathbb{R}^n \mid \forall i : a_i^\top p = 0\}$.

Next we show oppositely that $T_C(x) \supseteq \{p \in \mathbb{R}^n \mid \forall i : a_i^\top p = 0\}$. Similarly to before examine an arbitrary $p \in \{p \in \mathbb{R}^n \mid \forall i : a_i^\top p = 0\}$. Consider the function $z(\alpha) = x + \alpha p$ where $x \in \mathcal{C}$. For this choice of z we have $z(0) = x$ and $z'(0) = p$. We also have that $z(\alpha) \in \mathcal{C}$ since when inserting into the constraints:

$$\forall i : a_i^\top z(\alpha) + b_i = a_i^\top (x + \alpha p) + b_i = a_i^\top x + b_i + \alpha a_i^\top p \stackrel{*}{=} 0$$

The equality marked by $*$ follows from the fact that since $x \in \mathcal{C}$ then $a_i^\top x + b_i = 0$ and we have $\alpha a_i^\top p = 0$ by the definition of p . This shows that $T_C(x) \supseteq \{p \in \mathbb{R}^n \mid \forall i : a_i^\top p = 0\}$. We conclude the proof since:

$$T_C(x) \subseteq \{p \in \mathbb{R}^n \mid \forall i : a_i^\top p = 0\} \bigwedge T_C(x) \supseteq \{p \in \mathbb{R}^n \mid \forall i : a_i^\top p = 0\} \implies T_C(x) = \{p \in \mathbb{R}^n \mid \forall i : a_i^\top p = 0\} \quad \square$$

2 Experiments

Implementation and Ensuring Correctness: We implemented the algorithms exactly as shown in the lecture notes. As shown in section (1.1) we can use whether $\|(I - A^\top(AA^\top)^{-1}A)\nabla f(x_k)\| < \varepsilon$ as a stopping criterion. We used $\varepsilon = 0.001$. As a fail-safe we also stop if we reach a preset maximum number of iterations.

To ensure correctness of our implementations we tested with various initializations and random constraints to see if we could get ε convergence. Even though we did not achieve this on all functions our results seem to align with the results from assignment 1. Furthermore we tested to see if Newton's algorithm would still converge in a single step on f_1 which it did. As also stated on p. 57 of the lecture notes, any algorithm, that only proposes feasible steps, will for all k fulfill $Ax_k + b = 0$. Since this should be the case for our algorithms we also for each k asserted that $Ax_k + b = 0$ to ensure correctness.

Included functions and plots: We exclude f_2 from our analysis and experiments as it does not permit $d > 2$, and we'd like to experiment with a varying number of constraints (not just one or two). As do not know a way of analytically finding the global optimum of the constrained functions, we choose to generate plots with the stopping condition $\min_{\lambda} \|\nabla f(x_k) - A^\top \lambda\|$ on the y-axis instead of the usual $\|f(x_k) - f(x^*)\|$.

Generating random constraints $Ax + b = 0$: For a d dimensional optimization problem we fill matrix $A \in \mathbb{R}^{m \times n}$, $m \leq n$ with values $a_{i,j} \stackrel{\text{i.i.d.}}{\sim} \tan \mathcal{U}(-\pi/2, \pi/2)$. Our heuristic here is to uniformly sample the slopes along each dimension of the hyperplanes. To ensure that $\text{rank}(A) = m$, we compute the rank of A with `np.linalg.matrix_rank(A)`, and if it is less than m we try again. For the offsets we use $b \sim \mathcal{N}(0, 100^2)$.

Generating a feasible starting point: We generate random points, x , and then project them into the feasible region using

$$P(x) = x - A^\dagger(Ax + b) \in \mathcal{C}$$

To compute the pseudo inverse, A^\dagger , we use `np.linalg.pinv(A)`. To ensure that some of our generated constraints exclude the unconstrained optimum we simply insert the optimal argument, x^* , into our generated constraints and check that they are not satisfied.

The stopping criterion: As already mentioned we use whether $\|M\nabla f(x_k)\| < \varepsilon$ as a stopping criterion with $M := I - A^\top(AA^\top)^{-1}A$. We compute M once upon initialization of the algorithms and for the steepest descent algorithm this M is also used for "correcting" the usual step direction $p_k = -\nabla f(x_k)$ such that the corrected $p'_k = -M\nabla f(x_k)$ is feasible while still being as close to $-\nabla f(x_k)$ as possible. We use the same stopping criterion for Newton's algorithm even though M is here not directly used in the algorithm. M is still just an overhead calculation that is made once and since for $m \leq n$, M can be calculated in $O(m^2n) \subseteq O(n^3)$ time. This overhead does not change the time complexity of Newton's algorithm since the linear system solved in each step also requires $O(n^3)$ time.

Evaluating the Performance of the Algorithms: To do this we designed and executed some experiments. Firstly we as per usual measured and visualized how the algorithms converge towards their stopping criterion for 100 feasible initialization on each of the included functions (all except f_2). For this experiment we used 10 dimensionality with 5 constraints. Each function was evaluated across a set of 100 distinct and randomly generated starting points. The optimization algorithms were iterated until until the stopping criterion was satisfied, that is until the maximum iteration threshold of 100,000 was reached, or until attainment of the projected gradient norm below the designated epsilon threshold of 1e-4.

Secondly we did the same thing but now we varied the number of constraints to see how this number impacts the performance of the algorithms. We show 2, 4, 8, 16 and 24 linear constraints on a 32 dimensional optimization problem. The relevant results and visualizations are shown and discussed in the following section. For all experiments and runs we used $c_1 = 0.1$.

3 Results

See figure 1. We observe that `lincon-newton` exhibits a superior convergence rate for f_1 , f_4 , and f_5 compared to the steepest descent method. The steepest descent method outperforms Newton's method in finding the minimum

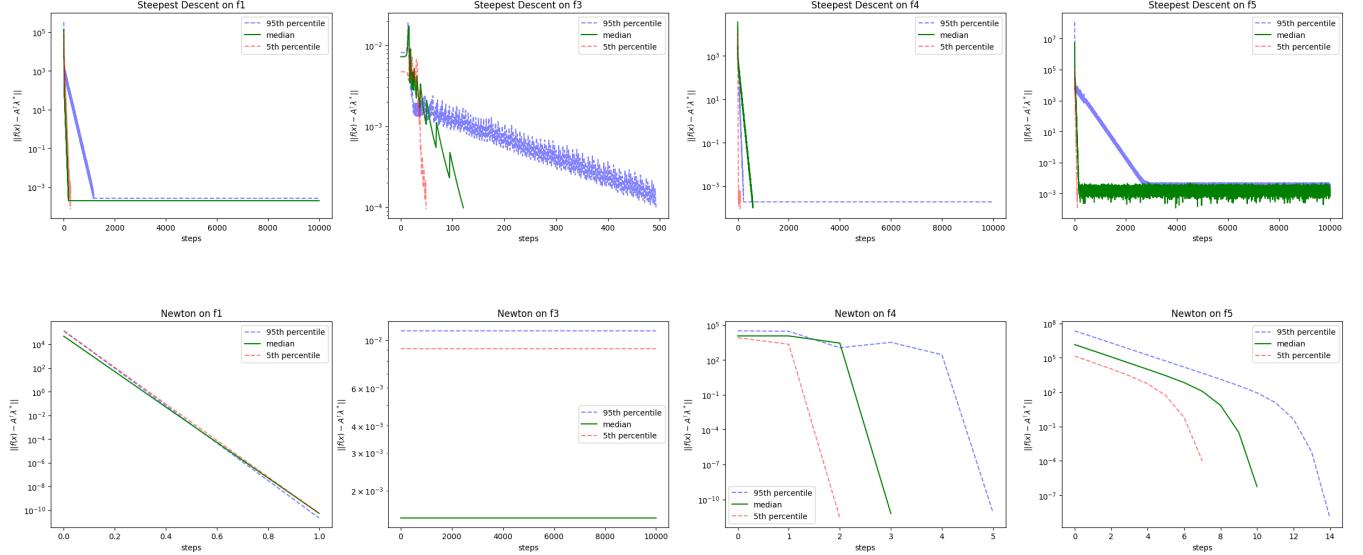


Figure 1: 10 dimensions, 5 constraints, $\epsilon = 1e-4$, 100 runs

for f_3 , given that Newton's method fails to converge on this function. We also observed this behaviour of Newton's in Handin 1, where it was also unable to converge on f_3 unless initialized close to the optimum.

The steepest descent method requires 500 iterations to identify the minimum in the worst cases. The stopping criterion (which is shown on the y-axis in these plots) jitters up and down; We hypothesize that this could be due to numerical issues when measuring the norm of high dimensional vectors. This becomes apparent especially in the steepest descent plots in this figure and figure 2, as steepest descent gets to take a lot more steps and the noise becomes visible, whilst `lincon-newton` converges quickly and the jittering is visible but has fewer steps to become apparent.

Now we'll cover figure 2. A general pattern we notice is that the more independent constraints we add, the faster the optimizer converges. Intuitively, this makes sense as the search space is reduced. Only with `lincon-newton` on f_4 this pattern breaks; we see almost the opposite ordering, but with a high variance. We don't consider the ordering significant and attribute it to random fluctuations; The difficulty of f_4 seems to be relatively less influenced by the number of constraints.

In both experimental evaluations, fluctuations in the projected gradient norm values were frequently observed. It was hypothesized that these fluctuations may have been attributed to the choice of the backtracking parameter c_1 in the optimization algorithm. However, attempts to investigate this hypothesis by varying the value of c_1 failed to yield any significant alterations in the observed results.

4 Conclusion

We have implemented and tested two optimizers able to handle linear equality constraints, `lincon-newton` and `lincon-steepest`. By the results of our tests, we deem both functional. As the optimizers will not necessarily be able to encounter a point x_k fulfilling $\nabla f(x_k) = \mathbf{0}$, we augment the stopping condition such that a gradient orthogonal the feasible space counts as a critical point. We also find that more constraints lead to faster convergence, at the cost of heavier computation.

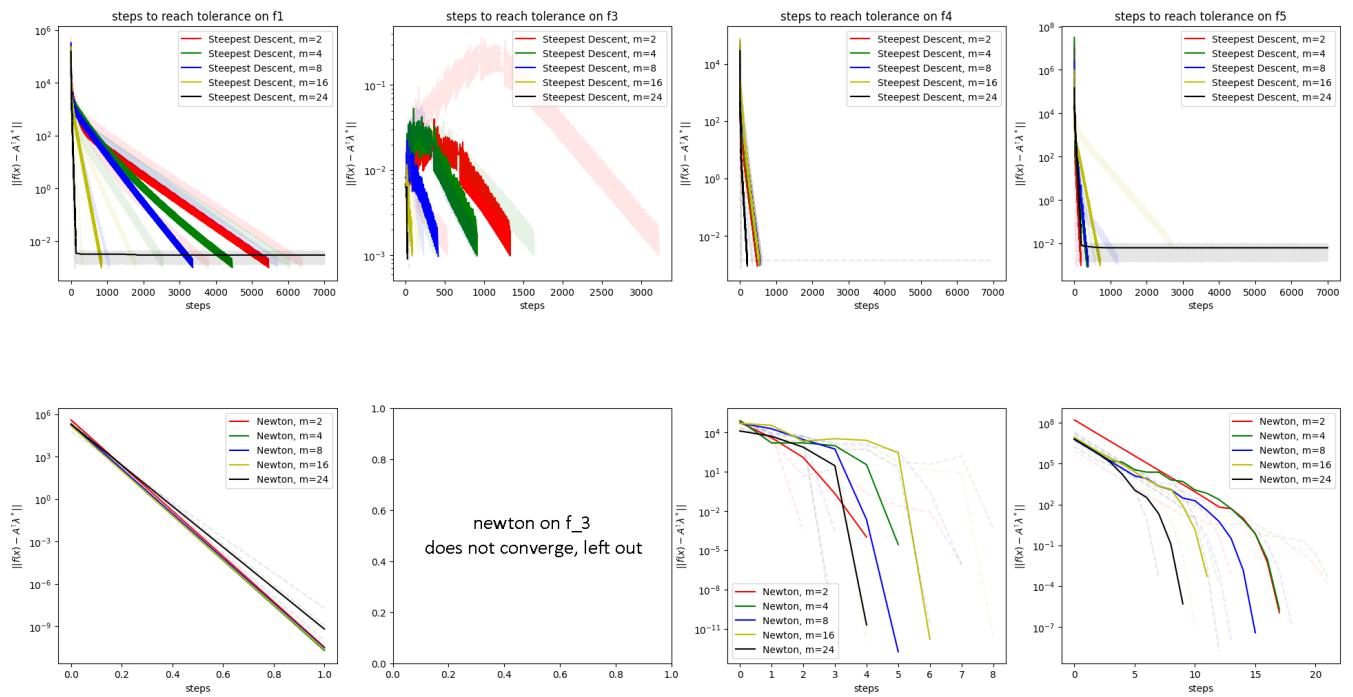


Figure 2: 32 dimensions, 2,4,8,16,24 constraints, $\epsilon = 1e - 3$, 20 runs