Numerical Optimization - Handin 2

This report seeks to investigate implementations of two algorithms, namely `Steepest Descent` and `Newtons Algorithm` using backtracking line search. We discuss the choice of stopping criterions for these algorithms, as well as how to pick the ρ and c_1 for the backtracking line search algorithm. Furthermore, an evaluation of whether the implementation is correct is done, and at last a discussion of what type of convergence rates the optimizers exhibit on the different functions.

1 Theory

1.1 Q-convergence of Newton on f_5

For $d = 2$ we have $f_5(\mathbf{x}) = x_1^2 + x_2^4$. The gradient and hessian are computed as:

$$\nabla f_5(x) = \begin{bmatrix} 2x_1 & 4x_2^3 \end{bmatrix}^\intercal$$
$$\nabla^2 f_5(x) = \begin{bmatrix} 2 & 0 \\ 0 & 12x_2^2 \end{bmatrix}$$

Since the hessian is a diagonal matrix the eigenvalues are simply the values in the diagonal. Assuming $x_{k,2} \neq 0$ all eigenvalues are positive and thus the hessian is positive-definite. This means the newton step direction is:

$$p_k = -(\nabla^2 f_5(\mathbf{x}_k))^{-1} \nabla f_5(\mathbf{x}_k) = -\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{12x_{k,2}^2} \end{bmatrix} \begin{bmatrix} 2x_{k,1} \\ 4x_{k,2}^3 \end{bmatrix} = \begin{bmatrix} -x_{k,1} \\ -\frac{x_{k,2}}{3} \end{bmatrix}$$

We now assume $\alpha_k = 1$ fulfills the sufficient decrease condition. In this case the newton step becomes:

$$x_{k+1} = x_k + p_k = \begin{bmatrix} x_{k,1} \\ x_{k,2} \end{bmatrix} + \begin{bmatrix} -x_{k,1} \\ -\frac{x_{k,2}}{3} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{2}{3}x_{k,2} \end{bmatrix}$$

The algorithm has Q-linear convergence since $r = \frac{2}{3} \in (0, 1)$ and:

$$\frac{||x_{k+1} - x^*||}{||x_k - x^*||} \overset{(x^* = 0)}{=} \frac{||x_{k+1}||}{||x_k||} = \frac{2}{3}\sqrt{\frac{x_{k,2}^2}{x_{k,1}^2 + x_{k,2}^2}} \leq \frac{2}{3}\sqrt{\frac{x_{k,1}^2 + x_{k,2}^2}{x_{k,1}^2 + x_{k,2}^2}} \overset{(x_{k,2} \neq 0)}{=} \frac{2}{3} = r$$

However the algorithm does not have Q-superlinear convergence since:

$$\frac{||x_{k+1} - x^*||}{||x_k - x^*||} = \frac{2}{3}\sqrt{\frac{x_{k,2}^2}{x_{k,1}^2 + x_{k,2}^2}} \overset{(x_{k,1} = 0)}{=} \frac{2}{3}\sqrt{\frac{x_{k,2}^2}{x_{k,2}^2}} \overset{(x_{k,2} \neq 0)}{=} \frac{2}{3} > 0$$

In the above case since the quotient $\frac{||x_{k+1} - x^*||}{||x_k - x^*||} \geq \frac{2}{3}$ we cannot find a sequence $\{r_k\}_{k \in \mathbb{N}}$ converging to 0 where r_k upper bounds the quotient for all x_k as r_k will inevitably become smaller than $\frac{2}{3}$.
The arguments above can be extended to $d > 2$. We will not include these extended argument explicitly but the idea is that:

$$\frac{||x_{k+1} - x^*||}{||x_k - x^*||} = \frac{\left|\left|\sqrt{\sum_{i=1}^d g(i,d) x_{k,i}^2}\right|\right|}{||x_k||} \quad ; \quad g(i,d) = \left(\frac{2(i-1)}{2i - 3 + d}\right)^2 \quad ; \quad \max_i g(i,d) = \left(\frac{2}{3}\right)^2 \quad ; \quad \min_{i \neq 1} g(i,d) > 0$$

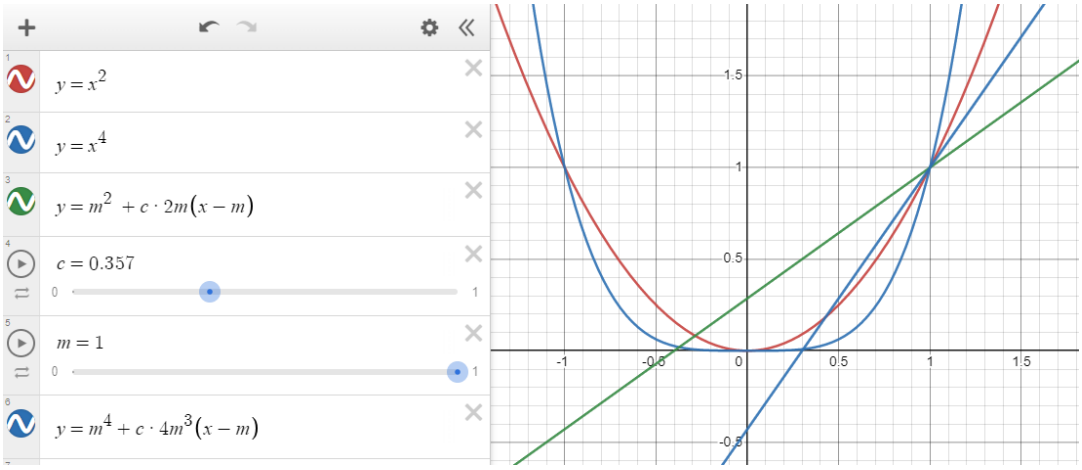

Figure 1: Ascreenshot from an interactive plot from Desmos, here's a link. If you're afraid of clicking it, it shows a value of $c_1 \approx 1/3$ for which the $x^* = 0$ fulfills the sufficient decrease condition from $x = 1$ on $f(x) = x^2$, but not on $f(x) = x^4$. To explain intuitively, the step will not be accepted because in comparison to how far the step is, the improvement is not good enough. The algorithm would rather stay within a local region with a lot of improvement.

To show Q-linear convergence we can upper bound the above by upperbounding $g(i, d)$ with $\max_i g(i, d)$ and similarly to lower bound the above to prove that it is not Q-superlinear convergence we just lowerbound $g(i, d)$ with $\min_{i \neq 1} g(i, d)$ and use our freedom to set $x_{k,i} = 0$ for all $i \neq d$ to show contradiction.

1.2 c_1 and the Sufficient Decrease Condition

The sufficient decrease condition for $f(x) = a \cdot x^m$, with $m = 2, 4, 6, ...$ where $\alpha_k = 1$ and $a, c_1 > 0$ can be manipulated to an equivalent inequality.
We choose $p_k = -x_k$.

$$
\begin{aligned}
& f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha_k p_k^\intercal \nabla f(x_k) \\
\iff & a(x_k - \alpha_k x_k)^m \leq a x_k^m - c_1 \alpha_k x_k a m x_k^{m-1} = a x_k^m - c_1 \alpha_k a m x_k^m \\
\iff & (x_k - \alpha x_k)^m \leq x_k^m - c_1 \alpha m x_k^m \\
\iff & x_k^m (1 - \alpha_k)^m \leq x_k^m (1 - c_1 \alpha_k m) \\
\iff & (1 - \alpha_k)^m \leq 1 - c_1 \alpha_k m \\
\iff & (1 - \alpha_k)^m \leq 1 - c_1 \alpha_k m \\
\iff & \frac{1}{1 - (1 - \alpha_k)^m} \frac{1}{m} \geq c_1 \\
\overset{(\alpha_k = 1)}{\iff} & \frac{1}{m} \geq c_1
\end{aligned}
$$

Above we observe that for $\alpha_k = 1$ the sufficient decrease condition is fulfilled if $c_1 \leq \frac{1}{m}$. For an interactive visualization of the consequences of this proof see figure 1.

2 Experiments

2.1 Stopping Criteria

For our stopping criteria, we make use of an `epsilon` parameter and a `max_iterations` parameter. If if our chosen convergence measure is below `epsilon` or the iteration counter exceeds `max_iterations` our algorithms stop. Our chosen convergence measure is the norm of the gradient. While we know the real optimum of each function we use

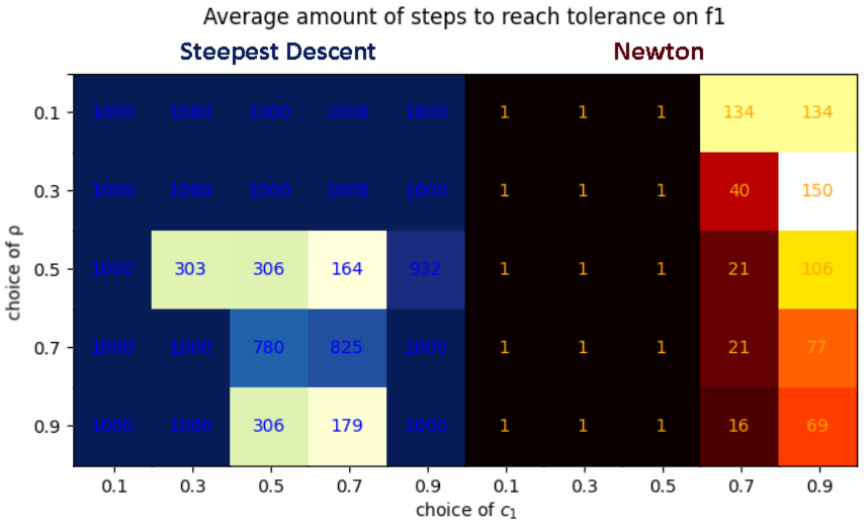

Figure 2: Heatmap of average steps until convergence for the two optimizers for different choices of the backtracking parameters c_1 and ρ. Here, 1000 is the maximum amount of steps before we abort and return.

the gradient norm to measure convergence to simulate "real-world" use. Furthermore we from lemma 1 of the lecture notes know that the algorithms theoretically are only guaranteed to converge to first order necessary conditions and thus measuring the gradient norm seems sensible. Using both `epsilon` and `max_iterations` allows us to balance the trade-off between accuracy and computational resources. The predetermined number of maximum iterations is a safety measure, since some algorithms under certain conditions may never terminate. Hence the `max_iterations` can be set to not exhaust our computational resources.
Picking the specific values for our stopping criteria parameters is done through trial and error. We want `max_iterations` large enough to give the algorithms enough time to converge, but we also need to take care of not setting it too high because sometimes the algorithms may converge intractably slow. As such, we have determined that a number of around 500 to 600 max iterations should suffice, while also making sure that our code runs at an acceptable speed. The value of `epsilon` is mainly a precision tolerance. In simple terms; *how close to the optimum is close enough?*. The visualized experiments have `epsilon` $= 0.01$ but this is somewhat arbitrary as we don't have a required accuracy of the algorithms. We tried a variety of `epsilon` setting but did not observe significant differences in the convergence patterns. Note that the iterations an algorithm require to converge can be highly dependent on the value of `epsilon`. If ϵ is decreased, then we would in some setting expect the required number of iterations for convergence to increase and thus `max_iterations` should be adjusted accordingly.

2.2 Picking values for backtracking line search

ρ should be chosen where $0 < \rho < 1$. In each iteration if the sufficient decrease condition is not fulfilled α is shrunk by a factor of ρ. Picking a value of $\rho = 0.5$ halves the step-length α of each iteration and when the sufficient step-length is found, it is at most a factor of 2 smaller than the previously tested step-length. With large ρ we may find a sufficient α value faster but one which is perhaps a lot smaller than the largest sufficient α and vice-versa with a small ρ. We heuristically do not want too small α since these are less likely to satisfy the curvature condition. c_1 should be chosen where $0 < c_1 < 1$. According to the lecture notes choosing c_1 fairly close to 0 (e.g. 0.1) makes the Wolfe conditions exclude steps that lead to bad or pathological behaviour as this c_1 accept many steps and removes the need for too much progress.[1]

To gauge the influence of the backtracking parameters, we explore 25 pairs of c_1 and ρ, taken from $\{0.1, 0.3, 0.5, 0.7, 0.9\}^2$. Figure 2 shows the average amount of steps (over 5 runs) until convergence for the different parameters. We choose the plot generated for f_1, which highlights the (in)sensitivity of `steepest descent` (`newton`). On the left half, two bright regions of assignments leading to a low amount of steps stand out, none of which include settings with a very

[1] Numerical Optimisation, Oswin Krause, February 5, 2023, page 22

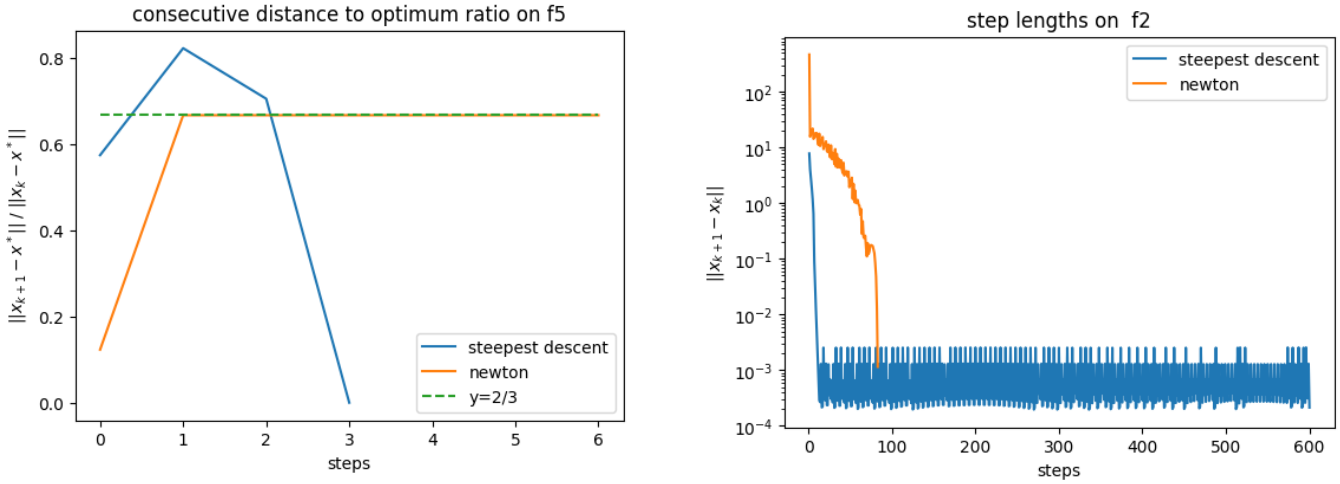

Figure 3: LEFT: Empirical observed ratio of the next over the current distance to the optimum, which is studied (analytically) when determining convergence speeds of optimizers on different functions. Here, we observe at most a ratio of $\frac{2}{3}$ on `newton`, which supports our previous analysis in the theory section. It also gives us faith in our implementation of the algorithm.
RIGHT: A plot showing `steepest descent`'s often erratic and slow progress, here on the Rosenbrock Banana function. `newton` also bounces around for a while, but the 2nd order information is used to take steps in a more fruitful direction, leading to fast convergence.

low c_1. From this we draw that `steepest descent` will overstep the optimum if not forced into some minimum amount of improvement. With too high c_1 though, steps are forced into being very short and too many are required to converge before we reach the max amount of steps. A low setting of ρ leads to erratic changes in step length, which will often significantly undershoot the optimal length. A higher ρ leads to more granularity in the considered values, but will also lead to potentially more step-shrinkages, which cost function evaluations.
`newton` is more robust to changes in the hyperparameters and performs the optimal step immediately in most of them. For high c_1 though, it starts having difficulties. As we saw in the theoretical exercise, the optimal value $x = 0$ will not necessarily fulfill the sufficient decrease condition, so we end up being forced into many small steps because of the strong curvature. A high c_1 can be somewhat remedied when paired with a high ρ, as `newton` ends up taking close to the longest step it can often, instead of shrinking the step to something tiny and taking many steps.

3 Correctness of our Implementation

To evaluate the correctness of our implemented algorithms we ran them for a handful of initializations on all functions and tested if they could get ϵ-close to the theoretical optima. We furthermore tested the correctness of our implementation of `newton` against our theoretical results from section 1.1. As seen in figure 3 the theory aligns with our empirical results. We also observed that our `newton` implementation converges in one step on f_1 which makes sense since an ellipsoid can be perfectly described by the second order approximation of `newton`.

4 Results/Discussion

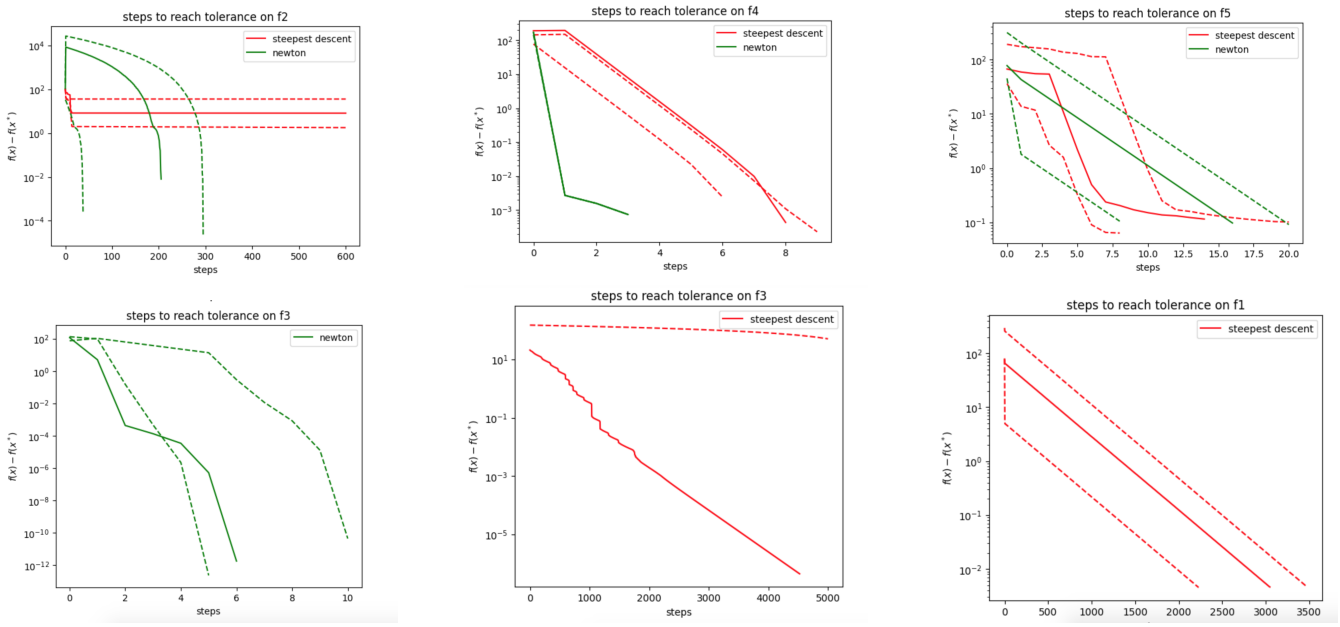

Figure 4: This figure shows 6 plots. It shows the number of steps to convergence on the different functions. The upper 3 plots shows convergence plots for both newton and steepest descent on function $\{f2, f4, f5\}$. The bottom three plots shows from the left: Convergence plot for newton on $f3$, in the middle we see convergence plot for steepest descent on $f3$ and last, we see convergence plot of steepest descent on $f1$ (Note that convergence plot for newton on $f1$ is omitted, since it always converges in the first step rendering it quite boring.)

	$f1$	$f2$	$f3$	$f4$	$f5$
Steepest Descent	Q-linear	Q-sublinear	Q-superlinear	Q-linear	Q-sublinear
Newton	Q-Quadratic	Q-quadratic or Q-superlinear	Q-superlinear	Q-Linear	Q-Linear

Table 1: Table consisting of what rates of convergence we believe the optimizers attain.

In figure 4 we see as denoted in the caption, our convergence plots measured in steps until convergence for each of the functions, where the runs with the `min`, `median` and `max` amount of steps until convergence are plotted. The values of our parameters for these experiments were: $\{\epsilon = 1e-2,\ c_1 = 0.5,\ \rho = 0.5\}$. What we see from the plot is that `newton` seems to beat `steepest descent` on all scales as far as our experiments can measure. It converges much quicker, and even in the case of $f1$ instantaneously solves the problem. `Steepest descent` seems to have some issues, where in some cases, even if we turned up the number of max iterations, it still did not converge. We can also see that the bottom middle and right plots have a much larger x-axis, driving home the claim that `steepest descent` is a lot less step efficient than `newton`.
In table 1 we show a table consisting of what rates of convergence we believe the different functions exhibit for both `steepest descent` and `newton`, based on figure 4. These are only guesses[2] however, as it needs to be determined analytically and not empirically (as it must hold for ALL sequences going to the minimum).

[2]Except for `newton` on f_5 which we have proven to be Q-linear in section **??**