Numerical Optimisation

Oswin Krause

March 1, 2023

On this Document

Structure We write proofs of statements in blocks

Proof. □

These blocks can be skipped on a first read through, even though we encourage the reader to work through them later on. Most proofs starts with an initial proof idea, which we highly recommend to read. The contents of some proofs can be referenced in later chapters.

Errors This document is version 1.5. Even though we checked derivations and text several times, there can be errors in the document and we might have to update the document several times throughout the curse. We are happy to correct any error you find.

Changelog: V1.1: fixed a number of typos in the appendix. Further fixed amissing factor of 1/2 in the taylor theorems used in Chapter 1.
v1.2: A number of typos fixed in Section 2. Most notably there were typos in Definition 3 and the last paragraph references the wrong entity.
v1.3: A number of typos fixed in Section 3.
v1.4: A number of typos fixed in Section 4. Notably: a sign error in eq (23) that only appeared there. v1.5: fixed a sign error in the definition of s_k in Algorithm 9.

Appendix: The appendix at the end covers a set of basic theorems, definitions and notations we require for this course. It is a good idea to quickly scroll over them to see whether you have an idea about what they mean. We will also refer to the theorems in the appendix explicitly the first time we use them in the material.

Contents

1 Basics of Numerical Optimisation

To optimise is to find the element in a set that achieves the best result according to some predefined quality measure. Optimisation is ubiquitous in almost every discipline, be it Mathematics, Physics, Economics, Chemistry, Biology, Computer Science and many many more areas. We all interact with the results of optimisation on a daily basis. The phones we carry are optimised for battery life, the parts they are made off are optimised to fit in a tiny chassis and the transistor layout of the CPU on the phone is optimised to require as little silicon as possible.

This course is about learning how to derive and implement optimisation algorithms. We will introduce optimisation algorithms that we analyse, implement as code, check for correctness and finally benchmark to find out which optimisation algorithm works best for which kind of task. Our goal is deep understanding of the methods. Rather than introducing a plethora of different optimisation algorithms, we will just introduce a maximum of one or two in each chapter, where we make sure that each algorithm offers something unique. We put focus on a detailed derivation, including the main intuitions and design criteria that lead to certain derivations.

In this course we will limit ourselves to continuous, constrained minimisation of a single objective function. We assume that we are given a set of solution candidates $\mathcal{C} \subseteq \mathbb{R}^n$ and an objective function $f : \mathcal{D} \to \mathbb{R}$ where we want to find the location of a minimum. The search space is usually a subset of the domain where the function is defined. Most often, $\mathcal{D} = \mathbb{R}^n$ and the function can be evaluated on all points outside the search space, but those solutions might not be desirable for certain reasons in the task.

In general, our goal is to solve the problem

$$x^* = \arg\min_{x \in \mathcal{C}} f(x) \ . \tag{1}$$

Minimisation is not a limitation, because if the goal of a task is to maximize f, then we can as well minimize $-f$. For this reason, when we talk about optimisation, we implicitly mean finding a minimizer of f. Often, finding the solution to problem (1) is difficult. While finding a point that is the minimum in some local area of the search space is often possible, there might be many such *local minima* in different areas and in general there is no efficient way to find the *global minimum*, which is the solution of (1). Thus, we are often content with finding a solution that is a local minimum.

If the search space $\mathcal{C} = \mathcal{D} = \mathbb{R}^n$ then we call the problem *unconstrained*: each possible argument of our vector space is a potential solution. This makes writing optimisation algorithms easier: when finding new candidates for an optimum, the algorithm does not need to ensure that they are element of $\mathcal{C}$. In contrast, if $\mathcal{C} \subsetneq \mathbb{R}^n$, then we call the problem *constrained*. In the first part of the course, we will be looking at unconstrained problems, before moving to constrained problems.

To solve an optimisation problem, we will introduce optimisation algorithms.

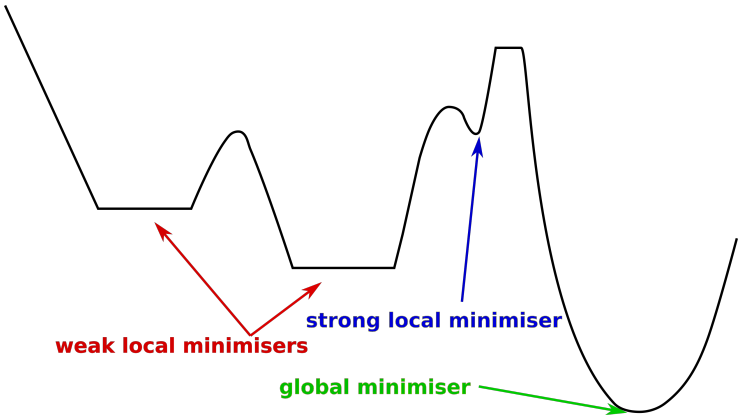


Figure 1: Visualisation of the three different classifications of minimisers.

Starting from an initial guess $x_0 \in \mathcal{C}$, that is often provided by the user, an optimisation algorithm is a set of rules that iteratively generates a set of candidate solutions $x_1, x_2, x_3, \cdots \in \mathcal{D}$ that are evaluated on the objective function to obtain function values $f(x_k)$ and possibly the gradient $\nabla f(x_k)$ or Hessian $\nabla^2 f(x_k)$ on the points x_k (For definition of gradient and Hessian, see appendix, Definition 14). Based on these values obtained in iterations $0, \ldots, k$, the algorithm will correct its internal estimate of the optimum as well as other information in iteration k to create a new guess x_{k+1}. As the algorithm progresses, the hope is that the function values become smaller and smaller, eventually ending up close to, or at a minimum of the function. In this case we say that the algorithm converges to a (local) optimum.

1.1 How do we recognize a local Minimum?

If our goal is to minimize, we have to be able to define what a minimum is and recognise when we find it. There are different classifications of minima, that we give in the following:

Definition 1 (Minimisers)**.** *When solving the unconstrained problem* (1) *with* $\mathcal{C} = \mathbb{R}^n$*, a point* x^* *is a*

1. *weak local minimiser of* f *if there is an open neighbourhood* $\mathcal{N} \subseteq \mathbb{R}^n$ *around* x^* *with*
$$f(x^*) \leq f(x),\ \forall x \in \mathcal{N}$$
2. *strict/strong local minimiser of* f *if there is an open neighbourhood* $\mathcal{N} \subseteq \mathbb{R}^n$ *around* x^* *with*
$$f(x^*) < f(x),\ \forall x \in \mathcal{N} \setminus \{x^*\}$$
3. *global minimiser of* f *if*
$$f(x^*) \leq f(x)\ \forall x \in \mathbb{R}^n$$

The difference of the three classifications can be seen in Figure 1. The first definition, the weak local minimiser, states that there exists an open set around x^* that does not include another point with lower function value. This area can be very small and there might be function values which are vastly better than this weak local minimiser outside of it. Moreover, there might be cases where there are multiple points that have the same function value, for example if the function has a plateau. If a minimiser is strict (or strong) then plateaus are ruled out: there must exist an open set around the local minimiser where all other points have a larger function value. Finally, we call a local minimiser global, if there is no better function value in the set of candidates $\mathcal{C}$. Minimisers can have several of these classifications at once. For example, every strong minimiser is also weak. For many functions, finding the global minimiser is a daunting task as there might be a large number of local minimisers and often there exist no additional information to navigate towards the global minimum. Thus, in minimisation, we are often content with an algorithm that finds a weak local minimum.

The definitions for local minima in Definition 1 are often complicated to work with. Especially for implementation in an algorithm, they can not be tested easily and require full knowledge about the function to define appropriate neighbourhoods and then show analytically that a point is the minimum in the set. In contrast, in an algorithm we can only query a finite number of points. Thus, we need more manageable ways to test whether a point is a minimum. This requires us to add additional assumptions on the function. For example, in the definitions above, we only needed the function value and the definitions hold for functions that are not differentiable or even discontinuous. Finding an efficient algorithm for functions like this is difficult, and often we add additional assumptions on the function itself, for example that it is differentiable around a local minimum. Under this condition, we are able to proof the following theorem:

Theorem 1 (First order necessary conditions)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable. If x^* is a weak local minimiser of f over a neighbourhood $\mathcal{N}$, then $\nabla f(x^*) = 0$.*

This theorem states that if we find a point that has gradient zero, this point is a possible candidate for a local minimum, and we refer to these points as *critical point* of f. However, while all local minima have gradient zero, this also holds for local maxima, and thus we need additional conditions. The proof of this theorem makes use of Taylor's Theorem.

Proof of Theorem 1. We show that we can find a better point in any neighbourhood around x^*, when $\nabla f(x^*) \neq 0$. We will use that $\nabla f(x^*) \neq 0$ to define a ray of points starting from x^* in a direction p. We will use this ray to construct points that have lower function value than x^* and show that we can construct such a point in any neighbourhood around x^*, which contradicts that x^* is a weak local minimum.

Assume that x^* is a weak local minimum and we have $\nabla f(x^*) \neq 0$. Let us define the vector $p = -\nabla f(x^*)$. Then, we have

$$p^T \nabla f(x^*) = -\nabla f(x^*)^T \nabla f(x^*) = -\|\nabla f(x^*)\|^2 < 0 \ .$$

Next, we define a function

$$g(\alpha) = f(x^* + \alpha p) - f(x^*) \ .$$

This function computes the function values of f along the ray $x^* + \alpha p$ and subtracts from it $f(x^*)$ such, that $g(\alpha) < 0$ when $f(x^* + \alpha p) < f(x^*)$. Careful differentiation of the function also reveals, that

$$g'(\alpha) = \nabla f(x^* + \alpha p)^T p \tag{2}$$

And thus, using the definition of p, we obtain

$$g'(0) = \nabla f(x^*)^T p = -\|\nabla f(x^*)\|^2 < 0 \ .$$

What we need to show now is that there exists a $u > 0$ such, that for all $0 < \alpha \leq u$ holds $g(\alpha) < 0$. In this case, any neighbourhood of x^* contains a point on the ray $x^* + \alpha p$, $0 < \alpha \leq u$ and thus, each neighbourhood contains a better point.
We will now create a first order Taylor expansion of g around 0. By Taylor's theorem (See appendix, Theorem (13)) this is

$$g(\alpha) = g(0) + g'(0)\alpha + R(\alpha) \ .$$

where the remainder term $R(\alpha)$ fulfils

$$\lim_{\alpha \to 0} \frac{R(\alpha)}{\alpha} = 0 \ ,$$

or more intuitively, that $|R(\alpha)| < h(\alpha)\alpha$, where $h(\alpha) \to 0$ as $\alpha \to 0$. This implies that there exists an u such, that $|g'(0)\alpha| > |R(\alpha)|$ for all $0 < \alpha \leq u$ and thus

$$g(\alpha) = \underbrace{g(0)}_{0} + \underbrace{g'(0)\alpha + R(\alpha)}_{<0} < 0, \forall 0 < \alpha \leq u$$

and this contradicts that x^* is a local minimiser. □

If the function is twice differentiable, we can make the conditions slightly stronger.

Theorem 2 (Second order necessary conditions)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable. If x^* is a weak local minimiser of f, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is a positive semi-definite matrix.*

Proof. We use the same strategy as in the previous theorem, just using second order information.

The condition that $\nabla f(x^*) = 0$ is true due to Theorem 1.
Assume for contradiction that $\nabla^2 f(x^*)$ is not positive semi-definite. Thus, there exists a vector $p \in \mathbb{R}^n$ such, that $p^T \nabla^2 f(x^*) p < 0$.
A second order Taylor expansion (see appendix, Theorem 14) around x^* gives

$$f(y) = f(x^*) + \underbrace{\nabla f(x^*)^T (y - x^*)}_{0} + \frac{1}{2}(y - x^*)^T \nabla^2 f(x^*)(y - x^*) + R(\|y - x^*\|) \ ,$$

where

$$\lim_{\alpha \to 0} \frac{R(\alpha)}{\alpha^2} = 0 \ .$$

We now set $y = x^* + \alpha p$ with $\alpha \in \mathbb{R}$ and this simplifies the last result to

$$f(x^* + \alpha p) = f(x^*) + \alpha^2 \frac{1}{2} \underbrace{p^T \nabla^2 f(x^*) p}_{<0} + R(\alpha \|p\|) \ .$$

Since by our assumption $p^T \nabla^2 f(x^*) p < 0$, there exists a $u > 0$ such, that $\alpha^2 |p^T \nabla^2 f(x^*)| < |R(\alpha \|p\|)|$ for all $0 < \alpha \leq u$ and thus $f(x^* + \alpha p) < f(x^*)$ for all $0 < \alpha \leq u$, which contradicts that x^* is a weak local minimiser. □

Finally, there are sufficient conditions that, if fulfilled, verify that a point is not only a weak local minimiser, but also strict.

Theorem 3 (Second order sufficient conditions)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable. Let be $x^* \in \mathbb{R}^n$ a point with $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then, x^* is a strict local minimiser.*

Proof. Our strategy is to construct a neighbourhood around x^* in which all other points have higher function value.

By second order Taylor expansion around x^*, we have

$$f(x^* + \alpha p) = f(x^*) + \alpha^2 \frac{1}{2} \underbrace{p^T \nabla^2 f(x^*) p}_{>0} + R(\alpha \|p\|) \ ,$$

where

$$\lim_{\alpha \to 0} \frac{R(\alpha)}{\alpha^2} = 0 \ .$$

The inequality $p^T \nabla^2 f(x^*) p > 0$ holds for any $p \in \mathbb{R}^n \setminus \{0\}$ as the Hessian

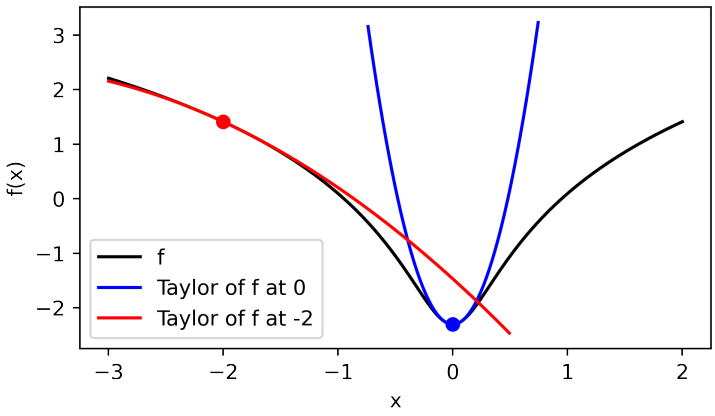


Figure 2: Visualisation of the second order Taylor expansion without remainder of some function f. When the Hessian has a negative eigenvalue (red curve) the function is curved downwards, while if it is positive (blue curve), the Taylor expansion is curved upwards.

is positive definite.
Therefore, there exists a $u(p) > 0$ such, that $\frac{1}{2}\alpha^2 p^T \nabla^2 f(x^*)p > |R(\alpha\|p\|)|$ for all $0 < \alpha < u(p)$.
As this holds for any p, we can define an open neighbourhood

$$\mathcal{N} = \{x^* + \alpha p \mid p \in \mathbb{R}^n, \|p\| = 1, 0 < \alpha < u(p)\} \ .$$

In this neighbourhood it holds, that $f(y) > f(x^*)$ for all $y \in \mathcal{N} \setminus \{x^*\}$, which shows that x^* is a strict local minimiser. □

It is important to note here that Theorem 3 is only one-way. There are strict local minimisers that do not fulfil it. An example for this is the function $f(x) = x^4$. The point $x^* = 0$ is a strict local minimiser, but as $\nabla^2 f(x^*) = 0$, we can not distinguish it from the saddle point of another function $g(x) = x^3$, which also fulfils the necessary second order conditions at x^*. This is because the Hessian provides curvature information, see Figure 2. If the Hessian is positive definite, then locally the function is curved upwards, or bowl-shaped in an area around the minimum. If the Hessian has eigenvalue 0 in some direction (or is completely 0 as in this 1D case), this means that the function looks locally flat like a plateau and we do not have any information whether it is curved up-or downwards around the critical point in this direction. Another result of this is that in many cases it is easier to find strict local minimisers than weak local minimisers.

For implementation in an algorithm, Theorems 1–3 are still not sufficient. This is because we can not expect an algorithm to ever locate the local mini-

mum exactly, and unless we do, the three theorems will correctly assert that the selected point is not a local minimum. Therefore, an algorithm that continues until it finds a point that fulfils any of these conditions, will run forever. Instead, we can make use of several heuristics that at the optimum agree with our theorems and around it present a measure of accuracy of the solution, which makes them candidates for *stopping conditions* of our algorithms.

The first and most important heuristic is to monitor the norm of the gradient. Instead of testing whether a candidate point x fulfils $\nabla f(x) = 0$, we can soften the requirement to $\|\nabla f(x)\| < \epsilon$, where ϵ is a chosen threshold value. This heuristic is based on the assumption of Theorem 1 that the objective function is continuously differentiable. In this case, as $x \to x^*$, we have that $\nabla f(x) \to 0$ and thus $\|\nabla f(x)\| \to 0$. If an algorithm converges to the optimum of a continuously differentiable function, no matter which value of ϵ we choose, it will eventually find a candidate point that fulfils the condition $\|\nabla f(x)\| < \epsilon$. This, however only holds in exact arithmetic. On a computer implementation, the limited precision of floating point numbers can prevent an algorithm from reaching the optimum if the chosen ϵ is too small.

There are other conditions. A common stopping condition is to limit the computation time or number of function evaluations of an algorithm. These are often stopgap solutions: either it is too difficult to accurately assess convergence, or there are real-world time limits and the best solution that can be found within that time has to be used instead. In some applications, or during benchmarking of algorithms, the optimal value $f(x^*)$ is known in advance and the algorithm is only needed to locate its position. In this case, the metric $f(x) - f(x^*) < \epsilon$ can be used to assess convergence for the same reasoning as before: as f is continuous, getting closer to the optimum means that the function value eventually decreases. However, this metric can still fail on functions with multiple minima, as the algorithm might get stuck on a worse local optimum and thus never reach the required tolerance. Finally, under benchmark conditions, we can also directly use $\|x - x^*\| < \epsilon$.

It is important to keep in mind that all these conditions are just heuristics, that can and will fail. For the conditions $\|\nabla f(x)\| < \epsilon$ and $f(x) - f(x^*) < \epsilon$ scaling of the function value is an issue, as we outline in the following with an example. Let $f(x) = ag(x)$, $a > 0$. Then

$$\begin{aligned} & \|\nabla f(x)\| < \epsilon \\ \leftrightarrow\ & \|\nabla a g(x)\| < \epsilon \\ \leftrightarrow\ & \|\nabla g(x)\| < \frac{\epsilon}{a} \end{aligned}$$

If we tried to optimise both f and g (with $a = 10^{-3}$) with an algorithm, then optimising f with $\epsilon = 10^{-8}$ stops earlier than when optimising g. Thus, the solution obtained on f will be less precise for a fixed ϵ.

There are also error conditions based on assumptions that are not fulfiled. Take for example the function $f(x) = |x|$ for $x \in \mathbb{R}$. This function is not differentiable at the optimum, $x^* = 0$, but outside of that point the gradient

can be computed and it is $\nabla f(x) = 1$ for $x > 0$ and $\nabla f(x) = -1$ for $x < 0$. Thus, for all $x \neq 0$ we have $\|\nabla f(x)\| = 1$.

Similar error conditions can be found for all heuristics. As another example, for the condition $\|x - x^*\| < \epsilon$ consider the following function:

$$f(x_1, x_2) = x_1^2 + 1000x_2^2$$

The optimum is $x^* = 0$ and we can look at two (x_1, x_2) points that fulfil the criterion: $(\epsilon, 0)$ and $(0, \epsilon)$. For the first point, we obtain $f(\epsilon, 0) = \epsilon^2$ and for the second $f(0, \epsilon) = 1000\epsilon^2$. We see that this heuristic ignores if an objective function is sensitive to small changes in a single argument and might return points with highly variable function values. In practice, these scaling differences can be massive. We commonly encounter problems where the scaling differences are a factor of 10^{10} or more.

Regardless of the stopping criterion used and their failure conditions, they are still important for the implementation of optimisation algorithms. Even if $\|\nabla f(x)\| < \epsilon$ prematurely stopped the algorithm at a saddle point of the function, for many algorithms this also means that we can not expect to make any more progress as the algorithm is stuck there. Thus, restarting the algorithm at a different location is the right thing to do.

1.2 Speed of Convergence to the Minimum

In practice, we are not only interested in whether the algorithm found a solution that fulfils our stopping criterion, but also visualize its progress. On the one hand, this helps us to find error conditions. If an algorithm did not reach the necessary precision of the solution before the maximum number of iterations was reached, we would like to analyse its progress and see whether running it for longer would have allowed it to reach the target eventually, or whether it is encountering numerical difficulties that make progress impossible. On the other hand, a visualization can be used to compare different algorithms to select the one that is best suited to solve a given task. While very simple, the metrics introduced before are all capable of comparing algorithm performance as we describe in the following.

We assume in the following that the algorithm is structured in a way as mentioned in the previous section: at each iteration, the algorithm generates a point x_k, $k = 1, 2, \ldots$ that is treated as a possible candidate for an optimum. Assuming the algorithm converges to the optimum as $k \to \infty$, this means that $x_k \to x^*$.

Different algorithms can converge to the optimum with different speed. Some algorithms can start slowly but as they build up knowledge about f they speed up, eventually outpacing other algorithms that start quicker and then slow down over time. This makes comparing algorithms difficult: if we execute the algorithms on a benchmark function for a short amount of time, their ranking will be different compared to longer runs. Thus, single numbers of final attained performance – independent of the metric used – are not indicative of the performance under different conditions. Finally, as the performance might be function

dependent, we also need a theoretical tools in order to analytically derive certain convergence speed properties.

We will classify algorithms based on their theoretical or observed convergence speed using the following scheme:

Definition 2. *Let $x_1, x_2, x_3, \dots$ be a sequence that converges to x^*. We say that the function converges*

Q-linearly *if there exists a $r \in (0,1)$ such, that*

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq r$$

Q-superlinearly *if there exists a sequence of $r_k \in (0,1)$, $r_k \to 0$ such, that*

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq r_k$$

Q-supralinearly *if there exists a sequence of $r_k \in (0,1)$, $r_k \to 1$ such, that*

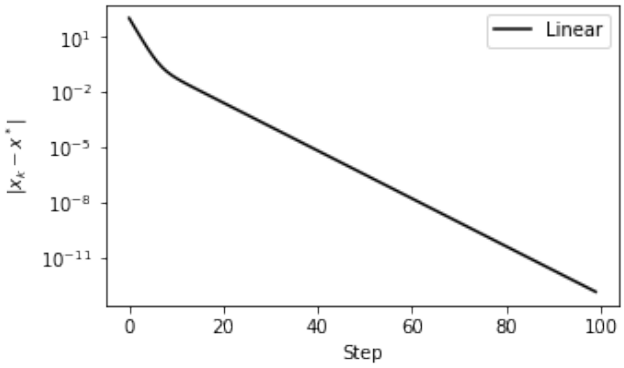
$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq r_k$$

Q-quadratically *if there exists an $M > 0$ such, that*

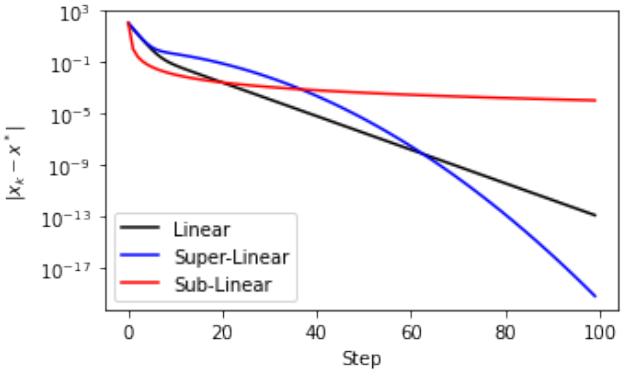
$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^2} \leq M$$

As an important remark: while we wrote the sequence in terms of $\|x_k - x^*\|$, any of the previous measures can be used instead. Intuitively Q-linear convergence means that in each iteration, the sequence reduces its distance from x^* by $100(1-r)\%$. Thus Q-linear convergence for an algorithm can already be very quick, as it amounts to convergence to the optimum with the speed of a geometric sequence. However, when $r \approx 1$, it can also take a long time to make any meaningful progress. Q-superlinear and Q-supralinear convergence are best understood in relation to Q-linear convergence. For a Q-superlinearly converging sequence, r_k is not constant, but steadily decreasing, which means that it makes more progress with every step. A Q-superlinear converging sequence will eventually converge faster than any Q-linear convergent sequence. For Q-supralinear convergence, this is reversed, here $r_k \to 1$ and thus it slows down. An example for this is the sequence $x_k = 1/k$, which converges to zero but at an ever slower rate.

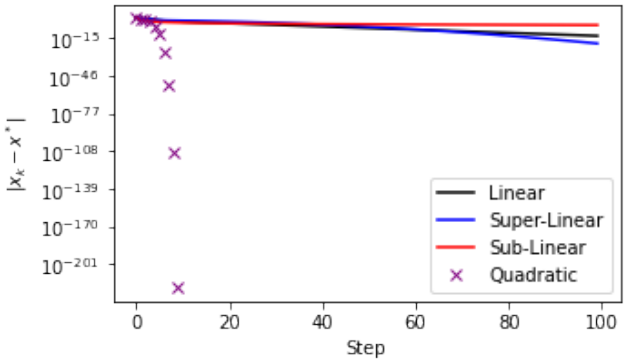
Finally, Q-quadratic convergence means an extremely rapid convergence. The best way to understand its speed is in terms on example: if $\|x_1 - x^*\| = 0.1$ and $M = 1$, then Q-quadratic convergence means that $\|x_2 - x^*\| < 10^{-2}$, $\|x_3 - x^*\| < 10^{-4}$, $\|x_4 - x^*\| < 10^{-8}$ and so on. So the number of digits in the exponent doubles with each step. In practice, it is difficult to differentiate Q-superlinear



(a) Q-Linear



(b) Q-Superlinear and Q-Supralinear



(c) Q-Quadratic

Figure 3: Examples of different sequences showing different speed of convergence. Q-quadratic is plotted as a set of points for better visibility. (TODO: change graphics)

from Q-quadratic convergence during an experiment, as the algorithm converges so rapidly to numerical precision that we do not have enough data to assess it.

It is easy to visualize Q-linear convergence, by using a semi-logplot where k has a linear axis and $\|x_k - x^*\|$ is plotted on log-scale. To show this we start by repeatedly applying the definition of Q-linear convergence to a sequence and obtain

$$\|x_{k+1} - x^*\| \leq r\|x_k - x^*\| \leq r^2\|x_{k-1} - x^*\| \cdots \leq r^k\|x_0 - x^*\| \ .$$

Thus, if we apply the logarithm on both sides and apply the logarithm rules, we get

$$\log\|x_{k+1} - x^*\| \leq k \log r + \log\|x_0 - x^*\| \ .$$

The right hand side has the form of an affine linear function in the number of iterations. Thus, on a semi-logarithmic plot, Q-linear convergence can be observed as a linear function, where the slope is $\log r$ and the offset is based on the initial point.

We have plotted sequences showcasing the different convergence properties in Figure 3. At first we show Q-linear convergence in Figure 3a. Here, at the beginning the sequence improves the value quickly before slowing down and eventually the iterates form a line. The slope obtained in the end can be used to measure convergence speed. That the sequence converges quicker at the beginning does not matter because Q-linear convergence allows that some steps make more progress, it just introduces a lower-bound to the amount of progress made. In Figure 3b we showcase both Q-superlinear and Q-supralinear convergence. The speedup or slowdown are clearly visible. Finally, Figure 3c shows Q-quadratic convergence. We see that at the beginning in the first few steps, convergence is slow but then progresses rapidly. Comparing the different heights of the marked iterates shows that each iteration approximately doubles the length of the previous step.

Visualisations like Figure 3 give more information on how quickly an algorithm converges: we can take the convergence measure, for example $\|\nabla f(x_k)\|$ and plot it on log-scale. Of course, just showing the plot does not proof that the algorithm has that type of convergence. For example, how do we know that an algorithm that initially exhibits Q-superlinear convergence will not eventually slow-down and fall back to superlinear convergence? This can not be done by an experiment and for that we need to analytically proof the convergence speed of algorithms.

1.3 Criteria for Good Algorithms

The choice of optimisation algorithm is in itself an optimisation problem. Ideally we would want to pick an algorithm that solves a given task reliably and quick and the obtained solution should be (approximately) correct. To be more exact, common criteria for algorithms are:

Robustness An optimisation algorithm should perform well on a wide variety of problems within their class, no matter the starting point

Stability The implementation of the algorithm should be numerically stable

Resource Efficiency An algorithm should use the provided computational resources (cpu time, memory, cost of function evaluations) efficiently

Accuracy The algorithm should be able to provide a solution with high accuracy

These objectives necessary conflict. As we will see, faster algorithm are often less robust or more difficult to implement in a stable manner. Most importantly, we will see that algorithms that require only a few function evaluations to find an approximate solution often use more compute resources to compute their next estimate of the local optimum. Many criteria involve both theoretical properties of the algorithm but also practical implementation details. For example Stability or Resource efficiency require a proper implementation. However, some algorithms might be more difficult to implement numerically stable, or it is not possible at all due to inherent difficulties of the optimisation approach.

The limitation of the criterion of robustness to certain problem class is necessary. For a restrictive class of problems, we might be able to find specialised algorithms that score higher on stability, efficiency and accuracy, but will not work on problems that do not fulfil the assumptions added by the restrictions. At the same time, no matter the algorithm, we have no problem finding a function that will break it. Thus, the measures make only sense given a certain function class and we can only compare algorithms on problems that they are both developed for.

2 Linesearch Algorithms

Data: $x_0 \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$
Result: Estimate of local minimum x_{k+1} of f
for $k = 0, \ldots,$ *until any stopping condition is fulfilled* **do**
  Pick descent direction p_k;
  Find α_k such, that $f(x_k + \alpha_k p_k) \leq f(x_k)$;
  $x_{k+1} = x_k + \alpha_k p_k$;

Algorithm 1: General line-search structure

Linesearch algorithms are one of the oldest and most important classes of optimisation algorithms and we give a basic description in Algorithm 1. The idea of the algorithm is a direct application of the proof idea of Theorem 1. There, we used that at a point x_k the gradient $\nabla f(x_k) \neq 0$ to construct a ray $x_k + \alpha p_k$ along which we then search for the value of α that leads to a smaller function value. Once we have constructed such a point on the ray, we can take this point as our next best guess for the optimum and continue until some stopping criterion is met.

In the proof of Algorithm 1, we chose $p_k = -\nabla f(x_k)$ because this ensured that $\frac{\partial}{\partial \alpha} f(x_k + \alpha p_k) < 0$ for $\alpha = 0$. Intuitively, this means that the function is locally descending when travelling a small distance along the ray and we used this property to construct a better point. But the choice $p_k = -\nabla f(x_k)$ is not necessary for the proof, and in fact the proof would hold for any p_k such, that

$$p_k^T \nabla f(x_k) < 0 \ . \tag{3}$$

A direction that fulfils equation (3) is called a *descent direction*.

After picking a descent direction, we need to find a suitable steplength α_k. In Algorithm 1, we only required that the newly found point is just better than the current point, but in reality the situation is a bit more complicated. The optimal steplength is the minimiser along the ray,

$$\alpha_k^* = \arg\min_{\alpha > 0} f(x_k + \alpha p_k) \ .$$

But finding the optimum along the ray, or an approximation to it is often difficult. For most functions, the analytic solution is not available and we would need another numerical algorithm to find it, which takes several evaluations of f and its derivatives. However, putting a lot of effort into finding the optimum is not very efficient: even if we we found a step length for which holds $\frac{\partial}{\partial \alpha} f(x_k + \alpha_k p_k) = 0$, then with $x_{k+1} = x_k + \alpha_k p_k$ we have

$$0 = \frac{\partial}{\partial \alpha} f(x_k + \alpha_k p_k) = \nabla f(x_k + \alpha_k p_k)^T p_k = \nabla f(x_{k+1})^T p_k \ .$$

Intuitively, at the optimum along the ray, the gradient of f is orthogonal to our current search direction and its length can still be very large. Thus, if

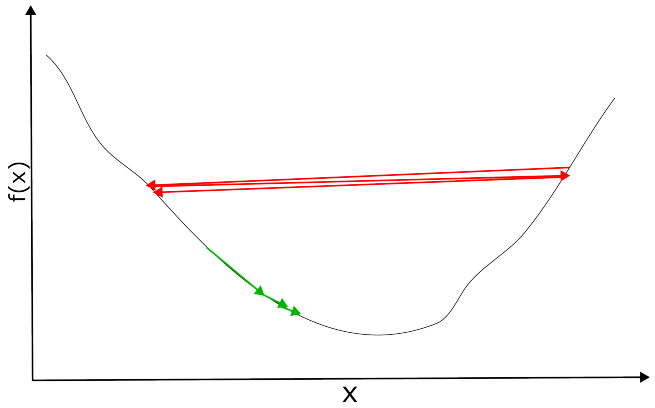


Figure 4: Two cases of wrongly chosen steplengths. Black: Graph of the function $f(x)$. Case 1 (red arrows): The steplength is chosen too long so that the algorithm makes barely any progress. Case 2 (green arrows): the step size shrinks too quickly such that the algorithm converges slower and slower towards the optimum.

we stopped earlier, we could have spent the additional function evaluations on exploring a different search direction.

When moving away from the optimal steplength, we still need a strategy to pick a sufficient α_k. Unfortunately, just choosing an arbitrary α_k with $f(x_k + \alpha_k p_k) < f(x_k)$ is not sufficient because we might only make very little progress. On the one hand, choosing only very small α_k can lead to slow progress towards the optimum and many iterations are needed to get close. If we are not careful, we might even converge prematurely as the steps become shorter too quickly to reach the optimum. On the other hand, a too long steplength might overshoot the optimum too far to make reasonable progress. Both examples are visualised in Figure 4. Many line-search algorithms therefore employ a steplength selection algorithm that ensure that certain conditions are met. The most common set of conditions are the *Wolfe Conditions*.

In the following we will first investigate steplength selection with the Wolfe conditions and show that under relatively weak conditions on the objective function and the chosen descent direction, the Wolfe conditions on the chosen steplength are enough to show that Algorithm 1 converges to a local optimum. Afterwards, we will introduce one example algorithm to choose the steplength and continue with two choices of descent directions, giving rise to the *Steepest Descent* and *Newton* algorithms.

2.1 Steplength Selection

Most algorithms need two conditions on the steplength to ensure convergence. We assume that we have selected a descent direction and search an $\alpha > 0$ that leads to improvement on the function when limited to the ray $x_k + \alpha p_k$, that is

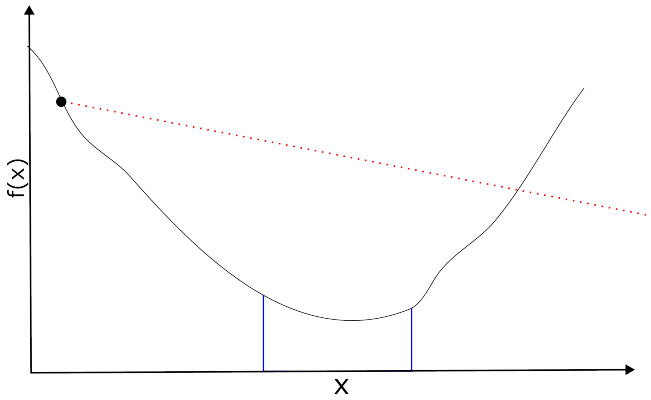


Figure 5: Visualisation of the Wolfe conditions. The black dot marks the current function value at $\alpha = 0$. The red dotted line visualises the sufficient decrease condition. Only steplengths with function value below that line are accepted. The blue lines mark the interval of steplength that fulfil the strong curvature conditions. Only derivatives with small absolute value are allowed.

finding an approximate minimum for

$$g(\alpha) = f(x_k + \alpha p_k) \ ,$$

with derivative

$$g'(\alpha) = \nabla f(x_k + \alpha p_k)^T p_k \ .$$

The Wolfe conditions use both quantities to decide whether a step length leads to sufficient progress for convergence. They are comprised of two conditions, a sufficient decrease condition and a curvature condition. The sufficient decrease condition states that

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha p_k^T \nabla f(x_k) \ , \tag{4}$$

where $0 < c_1 < 1$. We are visualizing the sufficient decrease condition in Figure 5. It is a stricter version than our initial condition stated in Algorithm 1 that $f(x_k + \alpha p_k) \leq f(x_k)$. It now requires that the new point is not only just better than the starting point, but by an amount that increases linearly with the steplength and in Figure 5 this means that the chosen step length must lead to a function value below the dotted line.

The sufficient decrease condition can be understood using Taylor's Theorem on g. When expanding g around 0, we obtain

$$\begin{aligned} & g(\alpha) = g(0) + \alpha g'(0) + R(\alpha) \\ \Leftrightarrow\ & f(x_k + \alpha p_k) = f(x_k) + \alpha \underbrace{p_k^T \nabla f(x_k)}_{<0} + R(\alpha) \end{aligned}$$

Again, the remainder R goes to zero as $\alpha \to 0$. If we take the first two terms, we can bound this by

$$f(x_k) + \alpha p_k^T \nabla f(x_k) < f(x_k) + c_1 \alpha p_k^T \nabla f(x_k), 0 < c_1 < 1 \ .$$

As $\alpha \to 0$ the remainder $R(\alpha_k)$ will shrink until eventually it is small enough so, that the sufficient decrease condition is fulfilled. This point always exists as the decrease by the linear term of the Taylor expansion is larger than the one required by the sufficient decrease condition. Thus, all step sizes small enough will fulfil it.

With a condition that prevents us from taking too long steps but allows us to take as small steps as we want, we need a second condition that prevents us from taking steps that are too short. We first begin by introducing the *strong curvature condition*, which is written as

$$|p_k^T \nabla f(x_k + \alpha p_k)| \leq c_2 |p_k^T \nabla f(x_k)| \ . \tag{5}$$

Again, $0 < c_2 < 1$ and this can be rewritten in terms of the first derivative of g, leading to

$$|g'(\alpha)| \leq c_2 |g'(0)| \ .$$

This means that the absolute value of the derivative at a candidate α must be sufficiently smaller than the value at the start, see Figure 5 for a visualisation. In combination with the sufficient decrease condition, the curvature condition can be weakened by removing the $|\cdot|$, which leads to the *curvature condition*

$$p_k^T \nabla f(x_k + \alpha p_k) \geq c_2 p_k^T \nabla f(x_k) \ . \tag{6}$$

This situation is visualised by Figure 5, when the second blue line is removed. Now all step lengths larger than the first blue line are accepted as the slope of the function values to the right side is positive. Both conditions together give rise to the Wolfe conditions

Definition 3 (Wolfe conditions)**.** *Let $x_k \in \mathbb{R}^n$, $\alpha > 0$ and $p_k \in \mathbb{R}^n$ so, that $\nabla f(x_k)^T p_k < 0$. If it holds that*

$$\begin{aligned} f(x_k + \alpha p_k) &\leq f(x_k) + c_1 \alpha p_k^T \nabla f(x_k) \\ p_k^T \nabla f(x_k + \alpha p_k) &\geq c_2 p_k^T \nabla f(x_k) \ , \end{aligned}$$

Then α is said to fulfil the Wolfe conditions. If further

$$|p_k^T \nabla f(x_k + \alpha p_k)| \leq c_2 |p_k^T \nabla f(x_k)| \ ,$$

then α fulfils the strong Wolfe conditions.

Assuming the Wolfe conditions hold at every step, we can proof convergence of many variations of Algorithm 1 to the optimum.

Lemma 1. *Consider any variation of Algorithm 1, where α_k satisfies the Wolfe conditions in each step. Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is bounded from below and continuous differentiable. Further, assume that the gradient of f is Lipschitz with Lipschitz constant L, i.e.,*

$$\|\nabla f(x) - \nabla f(x')\| \leq L\|x - x'\|, \text{ for all } x, x' \in \mathbb{R}^n$$

Then,

$$\sum_{k=0}^{\infty} \cos^2(\theta_k)\|\nabla f(x_k)\|^2 < \infty \ , \tag{7}$$

where

$$\cos(\theta_k) = -\frac{p_k^T \nabla f(x_k)}{\|\nabla f(x_k)\|\|p_k\|} \ .$$

Before we proof this theorem, a little bit of explanation and description of its importance is in order. Under conditions of the theorem, equation (7) implies that

$$\lim_{k\to\infty} \cos^2(\theta_k)\|\nabla f(x_k)\|^2 = 0 \ .$$

This means that as the algorithm progresses, either $\cos(\theta_k) \to 0$ or $\|\nabla f(x_k)\| \to 0$. In the latter case, we have that the algorithm approaches a point that fulfils the necessary first order conditions, Theorem (1), which is what we wanted. The first case is the cosine of the angle between p_k and $\nabla f(x_k)$. As p_k is a descent direction, this angle is between -90 and 90 degrees, which means that when $\cos(\theta_k) \to 0$ this is equivalent to p_k becoming orthogonal to $-\nabla f(x_k)$. This is something we have full control over, when picking a step and we can always ensure that $\cos(\theta_k) > \delta$. For example, if we pick $p_k = -\nabla f(x_k)$, then $\cos(\theta_k) = 1$. However, as we will see later, this is not the optimal choice.

Proof of Lemma 1. Proof Idea: As all steps fulfil the Wolfe conditions, we first use the curvature condition together with the Lipschitz assumption to derive a lower bound on α_k. We can then bound the single-step improvement of the function value $f(x_{k+1}) - f(x_k)$ using the sufficient decrease condition as a lower bound on the improvement as a function of the steplength, where larger steps give a better improvement in the lower bound. We then sum up all steps and use that f is bounded from below to argue that the sum of improvements is finite, leading to (7).

By assumption, we know that in each iteration an α_k exists that fulfils the curvature condition. By subtracting $p_k^T \nabla f(x_k)$ from both sides of the curvature condition (6), we obtain

$$p_k^T \left(\nabla f(x_k + \alpha_k p_k) - \nabla f(x_k)\right) \geq \underbrace{(c_2 - 1)}_{<0} \underbrace{p_k^T \nabla f(x_k)}_{<0} > 0 \ .$$

Where the last inequality holds since $c_2 < 1$ and p_k is a descent direction.

Similarly, we get using Cauchy-Schwarz inequality (See Appendix, Definition 12) and the Lipschitz condition, that

$$\begin{aligned} p_k^T \left(\nabla f(x_k + \alpha_k p_k) - \nabla f(x_k)\right) &\leq \|p_k\| \|\nabla f(x_k + \alpha_k p_k) - \nabla f(x_k)\| \\ &\leq \|p_k\| \left(\alpha_k L \|p_k\|\right) \\ &= \alpha_k L \|p_k\|^2 \end{aligned}$$

Combining the results, we obtain

$$\begin{aligned} (c_2 - 1) p_k^T \nabla f(x_k) &\leq \alpha L \|p_k\|^2 \\ \Leftrightarrow \alpha_k \geq \frac{c_2 - 1}{L} &\frac{p_k^T \nabla f(x_k)}{\|p_k\|^2} \end{aligned}$$

Since α_k also fulfils the sufficient decrease condition, we can insert the previous result in (4) and obtain

$$\begin{aligned} f(x_k + \alpha_k p_k) &\leq f(x_k) + c_1 \alpha_k p_k^T \nabla f(x_k) \\ &\leq f(x_k) - \underbrace{c_1 \frac{1 - c_2}{L}}_{c} \frac{p_k^T \nabla f(x_k)}{\|p_k\|^2} p_k^T \nabla f(x_k) \\ &= f(x_k) - c \frac{(p_k^T \nabla f(x_k))^2}{\|p_k\|^2} \\ &= f(x_k) - c \frac{(p_k^T \nabla f(x_k))^2}{\|p_k\|^2 \|\nabla f(x_k)\|^2} \|\nabla f(x_k)\|^2 \\ &= f(x_k) - c \cos^2(\theta_k) \|\nabla f(x_k)\|^2 \end{aligned}$$

Where the last step holds true, due to the definition of $\cos(\theta_k)$.
Due to the update equation of Algorithm 1, we have that $x_{k+1} = x_k + \alpha_k p_k$ and thus

$$f(x_{k+1}) \leq f(x_k) - c \cos^2(\theta_k) \|\nabla f(x_k)\|^2 \ .$$

We can now apply this result recursively and obtain

$$f(x_{k+1}) \leq f(x_0) - c \sum_{j=0}^{k} \cos^2(\theta_j) \|\nabla f(x_j)\|^2 \ .$$

Since f is bounded from below, we have that $|f(x_{k+1}) - f(x_0)| < \infty$ as $k \to \infty$ and therefore

$$\sum_{j=0}^{\infty} \cos^2(\theta_j) \|\nabla f(x_j)\|^2 < \infty$$

□

In Lemma 1 we assumed that in every step an α_k exists that fulfils the Wolfe conditions. Following the intuition of the curvature condition representing a lower bound and the sufficient decrease condition representing an upper bound on α_k, there is a-priori no guarantee that this is the case. We will show in the following that there exist values for c_1 and c_2 that work under the same mild assumptions as in the previous proof.

Lemma 2. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is bounded below and continuous differentiable. Let $x_k \in \mathbb{R}^n$ and $p_k \in \mathbb{R}^n$ so, that $\nabla f(x_k)^T p_k < 0$. Then, if $0 < c_1 < c_2 < 1$, there exist intervals of step lengths satisfying the Wolfe conditions and strong Wolfe conditions.*

Proof. The idea of this proof is that we first use the sufficient decrease condition to build an upper bound for steps to look for and then use the curvature condition to establish an interval inside this area that fulfils both conditions.
We again take $g(\alpha) = f(x_k + \alpha p_k)$ and further, let $l(\alpha) = f(x_k) + \alpha c_1 \nabla f(x_k)^T p_k$ the right hand side of the sufficient decrease condition (4). Since g is bounded from below and l is not, there exists a steplength $\alpha > 0$ for which $l(\alpha) = g(\alpha)$. Pick $\alpha_{\max}$ as the smallest such choice. For all $0 < \alpha < \alpha_{\max}$ the sufficient decrease condition holds.
Since f is continuous, we can apply the mean-value theorem (see appendix, Theorem 11). Thus, there exists $\alpha' \in (0, \alpha_{\max})$ for which holds

$$g(\alpha_{\max}) - g(0) = \alpha_{\max} g'(\alpha')$$

We can now insert this in our previous result:

$$\begin{aligned} & g(\alpha_{\max}) - g(0) = l(\alpha_{\max}) - g(0) \\ \Leftrightarrow \quad & \alpha_{\max} g'(\alpha') = \alpha_{\max} c_1 \nabla f(x_k)^T p_k \\ \Leftrightarrow \quad & g'(\alpha') = c_1 \nabla f(x_k)^T p_k \\ \Leftrightarrow \quad & \nabla f(x_k + \alpha' p_k)^T p_k = c_1 \nabla f(x_k)^T p_k \end{aligned}$$

This shows that $\nabla f(x_k + \alpha p_k)^T p_k < 0$ and since $c_1 < c_2$ and $\nabla f(x_k)^T p_k < 0$, we have that

$$\nabla f(x_k + \alpha' p_k)^T p_k = c_1 \nabla f(x_k)^T p_k > c_2 \nabla f(x_k)^T p_k \ .$$

Thus, α' fulfils the curvature condition and since $\alpha' \in (0, \alpha_{\max})$ it also fulfils the sufficient decrease condition. Moreover, since $\nabla f(x_k + \alpha' p_k)^T p_k < 0$ we have also shown that this point fulfils the strong Wolfe conditions.
Finally, since f is continuous, this also holds in a small interval around α', which concludes the proof. □

This completes the convergence proof of Lemma 1 as now we know that we

can choose a pair of c_1 and c_2 that ensures that steplengths exist that fulfil the Wolfe conditions. Most importantly, these values are independent of the function to optimise and thus, given a sufficient implementation of a steplength procedure, we can always find a step that leads to convergence. While the conditions $0 < c_1 < c_2 < 1$ are fairly broad, there are considerations that allow to pick suitable points. In practice, we want the Wolfe conditions to exclude steps that lead to especially bad or pathological behaviour. Thus, our goal should be to accept as many steps as possible. This means that c_1 should be fairly close to zero, as this removes the need for too much progress. Similarly, the curvature condition can be fairly close to one, but since it is directly related to the gradient convergence criterion, it can be used to enforce slightly more progress. Thus, a good default value is $c_2 = 0.9$.

Data: $x, p \in \mathbb{R}^n$, $0 < c_1 < 1$, $\alpha_{\text{init}} > 0$, $0 < \rho < 1$
Result: Steplength α that fulfils the sufficient decrease condition
$\alpha \leftarrow \alpha_{\text{init}}$;
while $f(x + \alpha p) > f(x) + c_1 \alpha p^T \nabla f(x)$ **do**
  $\alpha \leftarrow \rho\alpha$;

Algorithm 2: Backtracking Linesearch

In practice, developing an algorithm that fulfils the Wolfe conditions is not a trivial task and we will delay its introduction to Section 4. A simple heuristic is to only use the sufficient decrease condition and pair that with a strategy that initially proposes large steplengths that are shortened until a choice fulfils the conditions. This way the chosen steps tend to be large and thus the error condition that the curvature condition should protect from, too small steps, are unlikely. We present one such heuristic in Algorithm 2. The algorithm takes the information of the current iteration x_k and p_k and given three parameters c_1, $\rho < 1$ and $\alpha_{\text{init}} > 0$, a given step size is found. α_{init} is the assumed initial steplength. Then, the algorithm iteratively evaluates the point at the chosen steplength, and tests whether it fulfils the sufficient decrease condition. If it fails, the steplength is multiplied with ρ, which shrinks the steplength. This is continued until a sufficient step is found. For ρ we can again pick a suitable default. For example, with $\rho = 1/2$, the algorithm halves the step size at every step. As a consequence, the steplength found is a maximum a factor two smaller than the last steplength that did not fulfil the conditions and thus is unlikely to be too short. This of course depends on the choice of α_{init}, as here, a too short initial guess can hamper convergence. This initial value depends on the algorithm and should be picked with some care.

2.2 Step Direction Selection

The previous section covered how to pick a suitable steplength α_k given a chosen step direction p_k. In this section, we will introduce common strategies for finding a good point. Finding the optimal step p_k is as hard as finding a minimum of f. Thus, most approaches use a local approximation $m_k(p) \approx f(x_k + p)$ to compute

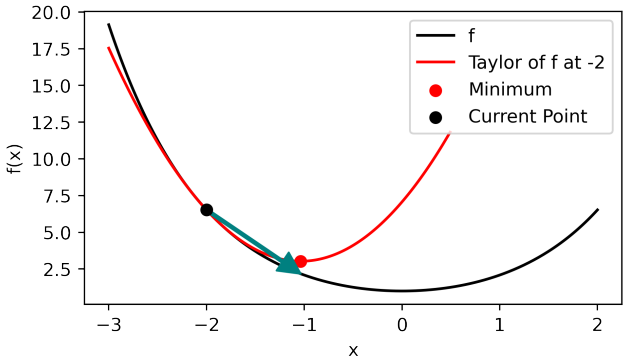


Figure 6: Visualisation of the Newton step in 1D. The model $m_k(p)$ (red) is the second order Taylor expansion of $f(x+p)$ at $x=-2$. The Newton step is then the point $x+p$ where p is the minimum of m_k. The step done by the optimiser is the teal arrow.

a step direction. This model should agree with the information that we have about f at x_k while it is still simple enough to compute analytic solutions. At a minimum, we want that

$$m_k(0) = f(x_k)$$
$$\nabla m_k(0) = \nabla f(x_k)$$

which means that our local approximation of f predicts the same function value and first derivative as f at the current point. The simplest model that fulfils this is the first order Taylor expansion without remainder:

$$m_k(p) = f(x_k) + p^T \nabla f(x_k) \tag{8}$$

By also requiring that the second derivative agrees, we arrive at the second order Taylor expansion without remainder as the model

$$m_k(p) = f(x_k) + p^T \nabla f(x_k) + \frac{1}{2} p^T \nabla^2 f(x_k) p \tag{9}$$

These models are two common examples. Ideally, if the model is an accurate approximation of f, then its prediction of a global minimum

$$p_k = \arg\min_p m_k(p) \tag{10}$$

is a descent prediction for a step. For the quadratic model (9), if the Hessian $\nabla^2 f(x_k)$ is positive definite, we can compute the minimum of problem (10) directly by solving the system of equations $\nabla m_k(p) = 0$. The solution is

$$p_k = -\left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k) \ . \tag{11}$$

As $\nabla^2 f(x_k)$ is positive definite, p_k fulfils the second order sufficient conditions, Theorem 3, on m_k. The step (11) is called the *Newton step*. We visualise it for a 1D example in Figure 6.

This approach does not work, if the Hessian $\nabla^2 f(x_k)$ has a negative eigenvalue. In this case, the Newton step is still a critical point of m_k, but it does not fulfil the second order necessary conditions, Theorem 2. The point p_k is therefore not a local minimum, but a saddle point or a local maximum[1]. The same holds for the linear model (8), which is monotonic decreasing and unbounded from below in the direction $p \propto -\nabla f(x_k)$ and thus no minimum exists.

For models that are not bounded from below, we can not produce a step by minimising the model value. Thus, we need other heuristics. For the linear model (8) we can investigate for which direction of p the problem descents quickest. This is the case for $p \propto -\nabla f(x_k)$ and thus, the usual direction taken is just

$$p_k = -\nabla f(x_k) \ . \tag{12}$$

This direction is called the *steepest descent* direction. However, this step direction provides no information about how far away the optimum of the function might be. The same approach does not work if the quadratic model (9) does not have a minimum, as the optimal step direction changes with the distance from the optimum. However, we can investigate changes to $\nabla^2 f(x_k)$ that make it positive definite. If we call this modified matrix B_k, then we can still compute a step

$$p_k = -B_k^{-1} \nabla f(x_k) \ .$$

At this point, we do not have to worry too much about how we achieve this, because as long as B_k is positive definite, we have

$$p_k^T \nabla f_k = -\nabla f(x_k)^T B_k^{-1} \nabla f(x_k) < 0 \ ,$$

and thus p_k is a descent direction.

2.3 Algorithms

We will now take the parts derived in the previous sections to create our first two optimisation algorithms. Since both algorithms will make use of the backtracking linesearch routine in Algorithm 2, we will refer to it in the algorithm as function `backtrack`$(x_k, p_k, \alpha_{\text{init}}, c_1, \rho)$ which takes the current point, step direction, initial step length and linesearch parameters c_1 and ρ and returns the final computed steplength α_k.

Steepest Descent With this, we begin with one of the simplest algorithms, Steepest Descent, which we present in Algorithm 3. For simplicity, we will refer to the choice of α_{init} in iteration k as β_k. As step direction, we use the steepest

[1] At this point it might make sense to go back to the proof of Theorem 2 and follow the steps using the quadratic function as an example to see how the negative eigenvalue affects the result

Data: $x_0 \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $0 < c_1 < 1$, $0 < \rho < 1$
Result: Estimate of local minimum x_{k+1} of f
$\beta_0 = 1$;
for $k = 0, \ldots,$ *until any stopping condition is fulfiled* **do**
  $p_k = -\nabla f(x_k)$;
  $\alpha_k = \text{backtrack}(x_k, p_k, \beta_k, c_1, \rho)$;
  $x_{k+1} = x_k + \alpha_k p_k$;
  $\beta_{k+1} = \frac{\alpha_k}{\rho}$;

Algorithm 3: Steepest Descent

descent direction (12), $p_k = -\nabla f(x_k)$. By Zoutendijk's Lemma, Lemma 1, this choice ensures convergence to the optimum, assuming α_k fulfil the Wolfe conditions, Definition 3. We then use the backtracking linesearch with initial length β_k to find α_k and then perform the step. This procedure ensures that the sufficient decrease condition is fulfiled.

The choice of β_k is important for two reasons. On the one hand, if β_k is too long, then we will spend a lot of function evaluations on shrinking it. In cases where the function is expensive to compute, this would slow down the algorithm significantly. On the other hand, if step sizes are chosen too small, they might not fulfil the sufficient decrease condition and we risk slow or premature convergence.

We solve both problems by picking $\beta_{k+1} = \alpha_k/\rho$. The parameter $0 < \rho < 1$ is used by the backtracking linesearch to decrease the steplength by factor ρ, when the sufficient decrease conditions are not fulfilled. Thus, if the new step length is too long, we only perform one additional step of backtracking. But if at some point our choice of steplength becomes too small, the backtracking linesearch will accept the proposed longer steplength and thus, over several iterations the step size will increase.

Unfortunately, it is difficult to proof better convergence properties than what we have done in Lemma 1. It is fair to say, that on most functions, Steepest Descent will eventually converge linearly as measured on $\|\nabla f(x_k)\|$. Most theoretical results however rely on exact linesearches, something we will seldom have in practice. However, we should not undersell the result in Lemma 1: Under the condition that the Wolfe condition holds, this theorem proofs *global* convergence, that means no matter the starting point Steepest Descent will always converge to a point fulfilling the first order sufficient conditions, even though it might sometimes take a long time.

Newton's Algorithm is probably the most well known fast optimisation algorithm. As we will show in Section 3, this algorithm has *local* quadratic convergence towards a strict local optimum. When the algorithm is started sufficiently close to the optimum it will converge at a quadratic rate. Most surprisingly, this holds even when choosing $\alpha_k = 1$.

The algorithm is based on choosing the step that minimises the quadratic

Data: $x_0 \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $0 < c_1 < 1$, $0 < \rho < 1$
Result: Estimate of local minimum x_{k+1} of f
for $k = 0, \ldots,$ *until any stopping condition is fulfilled* **do**
  if $\nabla^2 f(x_k)$ *is positive definite* **then**
    $p_k = -\left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)$;
  else
    Compute eigenvalues λ_i and eigenvector v_i of $\nabla^2 f(x_k)$;
    $H = \sum_{i=1}^{N} \frac{1}{|\lambda_i|} v_i v_i^T$;
    $p_k = -H \nabla f(x_k)$
  $\alpha_k = \text{backtrack}(x_k, p_k, 1.0, c_1, \rho)$;
  $x_{k+1} = x_k + \alpha_k p_k$;

Algorithm 4: Newtons Algorithm

model (9). As we have seen, this minimum exists if $\nabla^2 f(x_k)$ is positive definite and leads to the Newton step

$$p_k = -\left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k) \ .$$

If $f(x_k + p_k) \approx m_k(p_k)$, then $x_k + p_k$ is close to a minimum of f. This means that a step length of $\alpha_k = 1$ often works well. Unfortunately, this simple idea becomes more complicated as we can not guarantee that the model approximates f well for longer steps. This is why we still use the backtracking linesearch: if the point does not fulfil the sufficient decrease condition, we should decrease α_k until it does. As another complication, the Hessian might not be positive definite. In this case the computed critical point in the quadratic model is no longer a global minimum, but a saddle point, or even a local maximum, in which case we use a modified Hessian.

All these considerations lead to Algorithm 4. If the Hessian is positive definite, we try the Newton step and use the backtracking linesearch with $\alpha_{\text{init}} = 1$. Under best conditions, this will always accept the step with $\alpha_k = 1$, or shorten it to make sufficient progress.

When the Hessian is not positive definite, we modify its eigenvalues such that it becomes positive definite. We do this by replacing the eigenvalues λ_i of $\nabla^2 f(x_k)$ by their absolute value $|\lambda_i|$. The result is, that in directions of eigenvectors in which the original m_k was curved upwards, we obtain the exact behaviour of the Newton step. For directions in which m_k is curved downwards, this modification leads to conservative, cautious step lengths.

3 Inexact Newton & Conjugate Gradients

In this section, we want to dive a bit deeper into Newton's method, which is a simple, yet effective method. However, a practical implementation is complicated by indefinite Hessians. In this chapter, we will focus on a smaller class of functions to optimise. We now require for our objective function f that

$$\nabla^2 f(x) \text{ is positive definite, } \forall x \in \mathbb{R}^n \ .$$

In this case, the complications of indefinite Hessians can be removed from the algorithm as the Newton step always exists. Functions that have this property have the property that the first-order necessary conditions, Theorem 1, automatically imply the second-order sufficient conditions, Theorem 3. Moreover, it can be shown that on such functions, there is only one critical point. These functions are also called *strictly convex*.

To reiterate, the Newton step in iteration k of Algorithm 4 is the minimum of our local model

$$m_k(p) = \frac{1}{2} p^T \nabla^2 f(x_k) p + \nabla f(x_k)^T p \tag{13}$$

and the value is $p_k = -\nabla^2 f(x_k)^{-1} \nabla f(x_k)$. To compute this step, we have to compute the Hessian, invert it and then compute the Newton direction (or solve the system of equation $\nabla^2 f(x_k) p_k = \nabla f(x_k)$ for p_k). This can be a very expensive operation when the number of parameters become large. In fact, doubling the size of the problem increases the computation time of the Newton step by factor eight. Eventually, this will dominate the computation time of our algorithm and might make it impractical for larger problems. In this and the next chapter we will investigate ways to improve on Newtons step. In this section, we will focus on improving the computation time of computing the step, while in Section 4 we will also discuss ways to approximate the Hessian itself.

Still, before we do all this work, we need to show that Newton's algorithm is worth further investigation: we need convergence results. To start the analysis we are interested in computing conditions under which Algorithm 4 converges to the optimum. This will give us a global convergence result under relatively strong conditions. Then, we will derive a relation between the norm of the gradient and the step taken, which we proof in Lemma 3. From this, we will derive a local convergence result and show that Newtons algorithm will converge Q-quadratically to the optimum. Local convergence means that we have to choose an initial starting point that is close enough to the optimum. Surprisingly, this works with a step size $\alpha_k = 1$ and no steplength conditions is needed. With this result, we will then analyse Lemma 3 again from the angle of only taking an approximate Newton step. This will provide us with a similar gradient relation that depends on a parameter that governs the quality of approximation. This will allow us to trade-off computation time and convergence-speed. With this knowledge, we need an efficient algorithm that computes approximate Newton steps. We will introduce conjugate gradients as a way to iteratively compute

the solution. Finally, we bind everything together into an inexact Newton's method.

3.1 Convergence Properties of Newtons Method

We will now start with our analysis of Newtons method. To make a proof feasible, we will slightly change Algorithm 4 by assuming that our linesearch procedure finds a point that fulfils the Wolfe conditions, and not just the sufficient decrease conditions. This allows us to derive a global optimisation result:

Theorem 4. *Assume $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, the gradient is Lipschitz and $\nabla^2 f(x)$ is positive definite for all $x \in \mathbb{R}$. Further, the largest eigenvalue of $\nabla^2 f(x)$ is bounded from above and the smallest eigenvalue is bounded from below.*

Suppose we are given a sequence x_k, $k = 0, 1, \ldots$ computed by Algorithm 4 and assume additionally, that all α_k fulfil the Wolfe conditions. Then, the algorithm converges to a critical point.

Proof. Proof Idea: We will first show that the Conditions for Zoutendijk's Lemma (Lemma 1) hold and then use the bounds on the eigenvalues to show that the angle between gradient and the Newton step is bounded away from 90 degrees.

Since $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable and $\nabla^2 f(x)$ is positive definite, we have at every step

$$p_k = -\nabla^2 f(x_k)^{-1} \nabla f(x_k).$$

Since all steps fulfil the Wolfe conditions by assumption and ∇f is Lipschitz, we can apply Zoutendijk's Lemma and it holds:

$$\sum_{k=0}^{\infty} \cos^2(\theta_k) \|\nabla f(x_k)\|^2 < \infty \ ,$$

where

$$\cos(\theta_k) = \frac{p_k^T \nabla f(x_k)}{\|\nabla f(x_k)\| \|p_k\|} \ .$$

What remains is to show that $\cos^2(\theta_k)$ is bounded from below as in this case, the theorem implies

$$\|\nabla f(x_k)\| \to 0 \ .$$

To show this, let $\lambda_1(x)$ and $\lambda_N(x)$ be the smallest and largest eigenvalue of $\nabla^2 f(x_k)$, respectively. Since both are bounded by assumption, there exist constants $\lambda_{\min}, \lambda_{\max}$ such, that $0 < \lambda_{\min} \leq \lambda_1(x) \leq \lambda_N(x) < \lambda_{\max}$.

With these constants, we have

$$\|p_k\| = \|(\nabla^2 f(x_k))^{-1}\nabla f(x_k)\| \leq \left\|\frac{1}{\lambda_1(x)}\nabla f(x_k)\right\| \leq \frac{1}{\lambda_{\min}}\|\nabla f(x_k)\| \ .$$

The second step holds because $1/\lambda_1(x)$ is the largest eigenvalue of $\nabla^2 f(x_k)^{-1}$. Similarly, we can bound

$$\nabla f(x_k)^T p_k = -\nabla f(x_k)^T(\nabla^2 f(x_k))^{-1}\nabla f(x_k) \geq -\frac{1}{\lambda_{\max}}\|\nabla f(x_k)\|^2 \ .$$

By inserting these results, we can obtain a lower bound

$$\begin{aligned}\cos(\theta_k)^2 &= \frac{(p_k^T\nabla f(x_k))^2}{\|\nabla f(x_k)\|^2\|p_k\|^2} \\ &\geq \frac{1}{\lambda_{\max}^2}\frac{\|\nabla f(x_k)\|^4}{\|\nabla f(x_k)\|^2\|p_k\|^2} \geq \left(\frac{\lambda_{\min}}{\lambda_{\max}}\right)^2 > 0 \ .\end{aligned}$$

□

With this result, we have shown global convergence under strong conditions. However, convergence to the optimum alone does not tell us about the speed of convergence. As we are interested in convergence of the norm of the gradients, we need a relation that connects the quality of a step to the decrease in the gradient norm. The first step is provided by the next Theorem.

Lemma 3. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice differentiable and $m_k : \mathbb{R}^n \to \mathbb{R}$ as defined in (13). Then, it holds*

$$\|\nabla f(x_k + p)\| \leq \|Q(p)p\| + \|\nabla m_k(p)\|$$

where

$$Q(p) = \int_0^1 \nabla^2 f(x + tp) - \nabla^2 f(x)\, dt$$

Before we proof it, let us try to understand what it means. The right hand side consists of two terms. The second term is the norm of the gradient on our model ∇m_k at the chosen p. This is a quantity we can measure when we choose p. The first term is interesting, because it can be interpreted as the errors we make in our step due to the change of the Hessian of f compared to the Hessian of our model while travelling along the step direction p. If the Hessian only changes slowly, then $Q(p)$ will be small. However, if it changes quickly, then a long p might lead to large error.

Proof. The core idea of the proof is to introduce a way to rewrite the derivative $\nabla f(x_k + p)$ such, that it includes our model prediction of the step, as well as the Hessian. This can be done using the fundamental theorem of Calculus in order to turn a difference of gradients into an integral that involves the Hessian.

We will start by introducing the model into the gradient:

$$\begin{aligned}\|\nabla f(x_k + p)\| &= \|(\nabla f(x_k + p) - \nabla m_k(p)) + \nabla m_k(p)\| \\ &\leq \|\nabla f(x_k + p) - \nabla m_k(p)\| + \|\nabla m_k(p)\| \\ &\leq \|\nabla f(x_k + p) - \nabla f(x_k) - \nabla^2 f(x_k)p\| + \|\nabla m_k(p)\|\end{aligned} \tag{14}$$

To continue, we will use the fundamental theorem of calculus (see Appendix, Theorem 12), that states for any differentiable function g that

$$g(u) - g(0) = \int_0^u g'(t)\, dt \ .$$

We will apply this theorem by setting $g(u) = \nabla f(x_k + up)$ which has vector valued derivative

$$g'(u) = \nabla^2 f(x + up)p$$

and results in

$$g(1) - g(0) = \nabla f(x_k + p) - \nabla f(x_k) = \int_0^1 \nabla^2 f(x + tp)p\, dt$$

We insert this into (14)

$$\begin{aligned}&\|\nabla f(x_k + p) - \nabla f(x_k) - \nabla^2 f(x_k)p\| + \|\nabla m_k(p)\| \\ =&\left\|\int_0^1 \nabla^2 f(x + tp)p\, dt - \nabla^2 f(x_k)p\right\| + \|\nabla m_k(p)\| \\ =&\left\|\left(\int_0^1 \nabla^2 f(x + tp) - \nabla^2 f(x_k)\, dt\right)p\right\| + \|\nabla m_k(p)\|\end{aligned}$$

Which is the relation we wanted to proof. □

We can now use this result and apply it to the Newton step to obtain a relation on the possible convergence speed. Since the Newton step is the minimum of our model, the second term of Lemma 3 will be zero and we only need to bound the error introduced by the change of Hessian. To be able to obtain any results, however, we have to limit our class of functions further to bound how quickly the Hessian can change along a step.

Our proof will consider only a simplified version of Algorithm 4, in which $\alpha_k = 1$ and we further assume that the algorithm starts sufficiently close to the optimum. This turns our result into a *local* convergence result and it only applies to the full Algorithm 4, if we assume that the algorithm manages to get close enough to the optimum (for example on a function where Lemma 4 holds), and then once it is there the backtracking linesearch always accepts the Newton step. In this case the following theorem shows that, Newton's algorithm converges Q-quadratically.

Theorem 5. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable and assume that $\nabla^2 f(x^*)$ is positive definite. Further, assume that there exists an $L > 0$ such, that*

$$\|(\nabla^2 f(x+p) - \nabla^2 f(x))q\| \leq L\|p\|\|q\|, \ \forall p, q \in \mathbb{R}^n$$

Consider the iteration $x_{k+1} = x_k + p_k$, $p_k = -\nabla^2 f(x_k)^{-1}\nabla f(x_k)$.

If the starting point x_0 is chosen sufficiently close to a critical point x^* of f, then $(\|\nabla f(x_k)\|)_k \to 0$ Q-quadratically.

The proof will rely on a non-trivial identity of the euclidean norm of a vector-valued integral. For any function $g : \mathbb{R} \to \mathbb{R}^n$ with g_i absolute integrable, and some interval (a, b), we have

$$\left\| \int_a^b g(x)\, dx \right\| \leq \int_a^b \|g(x)\|\ dx \ .$$

This can be shown using properties of the euclidean norm and is left as an exercise.

Proof of Theorem 5. The proof is a direct application of Lemma 3. Since the Hessian is positive definite on the full domain, the Newton step p_k exists and since p_k is the global optimum of m_k, we have $\nabla m_k(p_k) = 0$. With this, we have

$$\begin{aligned}
\|\nabla f(x_{k+1})\| &= \|\nabla f(x_k + p_k)\| \\
&\leq \left\| \int_0^1 \nabla^2 f(x_k + tp_k) - \nabla^2 f(x_k)\, dt\, p_k \right\| \\
&\leq \int_0^1 \left\| \left(\nabla^2 f(x_k + tp_k) - \nabla^2 f(x_k)\right) p_k \right\| dt \\
&\leq \int_0^1 tL\|p_k\|^2\, dt \leq \frac{L}{2}\|p_k\|^2 \ .
\end{aligned}$$

Finally, we need to bound the length of the Newton step. We again define

with $\lambda_1(x)$ the smallest eigenvalue of $\nabla^2 f(x)$. With this, we arrive at

$$\begin{aligned}\|\nabla f(x_{k+1})\| &\leq \frac{L}{2}\|p_k\|^2 \\ &= \frac{L}{2}\|(\nabla^2 f(x_k))^{-1}\nabla f(x_k)\|^2 \\ &\leq \frac{L}{2\lambda_1(x_k)}\|\nabla f(x_k)\|^2 \ .\end{aligned}$$

We now need to find a proper starting point. Since $\nabla^2 f$ is continuous and positive definite at x^*, there exists an open neighbourhood $\mathcal{N}$ around x^* in which $\lambda_1(x) > \lambda_1(x^*)/2$, for all $x \in \mathcal{N}$. Thus, for $x_k \in \mathcal{N}$ it holds

$$\|\nabla f(x_{k+1})\| \leq \frac{L}{\lambda_1(x^*)}\|\nabla f(x_k)\|^2 \ . \tag{15}$$

We can now pick a starting point $x_0 \in \mathcal{N}$ such, that the right hand side of (15) is sufficiently smaller than one. In this case, one can show that the sequence never leaves $\mathcal{N}$ and thus we have that the sequence converges with convergence rate

$$\frac{\|\nabla f(x_{k+1})\|}{\|\nabla f(x_k)\|^2} \leq \frac{L}{\lambda_1(x^*)}$$

which concludes the proof. □

Finally, we want to shed a light on approximate Newton steps. We assume that instead of finding the analytical minimum of m_k, we use an optimiser that only finds a value close to the optimum. We will measure closeness in terms of relative size of the gradients and require that the computed p fulfils

$$\|\nabla m_k(p)\| \leq \eta_k \|\nabla f(x_k)\| \ .$$

In this case, the second term in Lemma 3 will not be zero, as we state more precisely below:

Lemma 4. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice differentiable and let $\lambda_1(x)$ and $\lambda_N(x)$ be the smallest and largest eigenvalue of $\nabla^2 f(x)$ and assume that $\lambda_1 > 0$. Further, assume that there exists an $L > 0$ such, that*

$$\|(\nabla^2 f(x+p) - \nabla^2 f(x))q\| \leq L\|p\|\|q\|, \ \forall p, q \in \mathbb{R}^n$$

Consider the iteration $x_{k+1} = x_k + p_k$, where p_k is chosen such, that $\|\nabla m_k(p_k)\| \leq \eta_k\|\nabla f(x_k)\|$, $0 \leq \eta_k < 1$. The sequence fulfils

$$\|\nabla f(x_{k+1})\| \leq \frac{L}{2}\left(\frac{1}{\lambda_1(x)} + \lambda_N(x_k)\eta_k\right)^2 \|\nabla f(x_k)\|^2 + \eta_k\|\nabla f(x_k)\| \ .$$

The first term still shows quadratic improvement in $\|\nabla f(x_k)\|$ and can therefore allow for rapid convergence, when the sequence is close enough to the optimum (albeit, the prefactors could be very large). The second term is only linear in $\|\nabla f(x_k)\|$ and thus the choice of η_k affects convergence speed. If we pick η_k as a constant, then the best we can expect to achieve is linear convergence. However, if we let η_k decay, for example

$$\eta_k = \min\left\{\frac{1}{2}, \|\nabla f(x_k)\|\right\} \quad ,$$

Then the second term also depends quadratically on $\|\nabla f(x_k)\|$. This gives us a degree of freedom to trade-off saving computation time in computing the Newton step with the number of steps until we reach the desired precision. For example, using

$$\eta_k = \frac{1}{2}\min\left\{\frac{1}{2}, \sqrt{\|\nabla f(x_k)\|}\right\} \quad ,$$

allows for superlinear convergence, while not requiring too many solver steps.

Proof of Lemma 4. Let $q_k = -\nabla^2 f(x_k)^{-1}\nabla f(x_k)$ be the Newton step. Inserting into Lemma 3 leads to

$$\|\nabla f(x_k + p_k)\| \leq \|Q(p_k)p_k\| + \|\nabla m_k(p_k)\| \tag{16}$$

Where $Q(p)$ is defined as in Lemma 3. For the first term, we can now follow similar derivations as in the previous proof and the triangle inequality to arrive at

$$\begin{aligned}\|\nabla f(x_k + p_k)\| &\leq \frac{L}{2}\|p_k\|^2 + \|\nabla m_k(p_k)\| \\ &\leq \frac{L}{2}\left(\|p_k - q_k\| + \|q_k\|\right)^2 + \|\nabla m_k(p_k)\| \end{aligned} \tag{17}$$

Next, we get a bound on $\|p_k - q_k\|$. We use

$$\begin{aligned}\|\nabla m_k(p_k)\| &= \|\nabla f(x_k) + \nabla^2 f(x_k)p_k\| \\ &= \|\nabla^2 f(x_k)(\nabla^2 f(x_k)^{-1}\nabla f(x_k) + p_k)\| \\ &= \|\nabla^2 f(x_k)(p_k - q_k)\| \geq \frac{1}{\lambda_N(x_k)}\|p_k - q_k\|\end{aligned}$$

Now, including our assumption on p_k, we have bounded

$$\frac{1}{\lambda_N(x_k)}\|p_k - q_k\| \leq \|\nabla m_k(p_k)\| \leq \eta_k\|\nabla f(x_k)\|$$

and from this follows:

$$\|p_k - q_k\| \leq \lambda_N(x_k)\eta_k\|\nabla f(x_k)\|$$

Finally, we need a bound on $\|q_k\|$, which we can bound using the smallest eigenvalue of $\nabla^2 f(x_k)$as in the previous proof:

$$\|q_k\| \leq \frac{1}{\lambda_1(x_k)} \|\nabla f(x_k)\|$$

Inserting everything in (17) and simplifying, we obtain the end result. □

3.2 Conjugate Gradients

In the last section, we have discussed how we can trade-off computation time of computing the Newton step with convergence on the function. This of course requires a good algorithm for approximately computing the Newton step that allows to make this trade-off efficiently. In this section, we will introduce the conjugate gradient algorithm, which is specialized to minimizing quadratic functions of the form

$$f(x) = \frac{1}{2} x^T Q x + g^T x \ .$$

Since the model m_k (13) from which we derived the Newton step has this exact form, we can use the conjugate gradients algorithm to find the minimum. We will not give a full derivation of the algorithm, but introduce its most important concepts.

The algorithm is based on the concept of *conjugancy*. Two vectors, $p, q \in \mathbb{R}^n$ are called conjugate with respect to a positive definite matrix Q, if it holds

$$p^T Q q = 0 \ .$$

Conjugancy is a generalization of orthogonality, as two orthogonal vectors are conjugate wrt to the matrix I_n. Indeed, one can show that there can be at most n conjugate vectors, which form a basis. We will make use of this property in the following.

Before we begin, let us investigate how a linesearch works on a quadratic function. So far, we have assumed that we can not compute the optimal steplength for a chosen direction. However, since the function we optimise is a known quadratic – we know the values of the parameters Q and g – we can compute the optimal step length α_k:

$$\begin{aligned} & \frac{\partial}{\partial \alpha_k} f(x_k + \alpha_k p_k) = 0 \\ \Leftrightarrow & (x_k + \alpha_k p_k)^T Q p_k + g^T p_k = 0 \\ \Leftrightarrow & \qquad \alpha_k = -\frac{x_k^T Q p_k + g^T p_k}{p_k^T Q p_k} \end{aligned} \tag{18}$$

The combination of using conjugate search directions and using an exact linesearch together allow for very efficient optimisation algorithms on quadratic functions, as the following result shows:

Theorem 6. *Let $f(x) = \frac{1}{2}x^TQx + g^Tx$, Q symmetric positive definite and let p_i, $i = 0\ldots, n-1$ be pairwise conjugate with respect to Q. Consider the iteration $x_{k+1} = x_k + \alpha_k p_k$, where α_k is computed via (18). It holds*

$$\nabla f(x_{k+1})^T p_i = 0,\ i = 0, \ldots, k \tag{19}$$

and we have $\nabla f(x_n) = 0$.

Proof. The last part of the statement follows directly from equation (19), since for $k = n-1$, $\nabla f(x_n)$ is orthogonal to the n direction vectors p_i. As these vectors form a basis due to conjugancy, this means that $\nabla f(x_n) = 0$. All that remains is to proof (19) by induction.

Let $x_0 \in \mathbb{R}^n$ be an arbitrary point. At iteration $k = 0$, (19) holds by construction as

$$\nabla f(x_1)^T p_0 = \nabla f(x_0 + \alpha_0 p_0)^T p_0 = \frac{\partial}{\partial \alpha_0} f(x_0 + \alpha_0 p_0) = 0$$

Now assume that (19) holds for an arbitrary $k > 0$ and thus

$$\nabla f(x_k)^T p_i = 0,\ i = 0, \ldots, k-1\ .$$

We have that

$$\nabla f(x_{k+1}) = Qx_{k+1} + g = Qx_k + \alpha_k Qp_k + g = \nabla f(x_k) + \alpha_k Qp_k$$

and thus for p_i, $i < k$ we have

$$\nabla f(x_{k+1})^T p_i = \underbrace{\nabla f(x_k)^T p_i}_{0} + \alpha_k \underbrace{p_k^T Q p_i}_{0}\ ,$$

where the first term is zero due to the assumption of the induction step, and the last term is zero due to conjugancy. Lastly, we have

$$\nabla f(x_{k+1})^T p_k = \frac{\partial}{\partial \alpha_k} f(x_k + \alpha_k p_k) = 0$$

due to construction of α_k. This completes the proof by induction. □

If we manage to construct conjugate directions cheaply, we have a fast iterative method, that we can stop, as soon as $\|\nabla f(x_{k+1})\|$ is smaller than some chose error tolerance ϵ. The naive approach is to use Gram-Schmidt orthogonalisation. Let us assume we have a given vector v_{k+1} and we want to generate from it a vector p_{k+1} that is conjugate to $p_0, \ldots, p_k$ that are each pairwise conjugate.

Since all p_i are independent, we can write

$$v_{k+1} = p_{k+1} + \sum_{i=0}^{k} \beta_i p_i \ ,$$

where $\beta_i \in \mathbb{R}$ and $p_{k+1} \in \mathbb{R}$ is the part that is conjugate to $p_0, \dots, p_k$. Our goal is now to extract the conjugate part p_{k+1} from v_{k+1}. With our definition of v_{k+1}, we have for $i = 0, \dots, k$ due to conjugancy that

$$v_{k+1}^T Q p_i = p_{k+1}^T Q p_i + \sum_{j=0}^{k} \beta_j p_j^T Q p_i = \beta_i p_i^T Q p_i \ ,$$

and thus

$$p_{k+1} = v_{k+1} - \sum_{i=0}^{k} \underbrace{\frac{v_{k+1}^T Q p_i}{p_i^T Q p_i}}_{\beta_i} p_i \ . \tag{20}$$

This approach allows us to extract p_{k+1}, but it is expensive: in the last step, the operation requires n matrix-vector multiplications, which has almost the same cost as matrix inversion via Gaussian elimination. Thus, it does not appear at first that this is an efficient algorithm. Luckily, for a specific choice of v_{k+1} in each iteration, orthogonalisation is cheap. It can be shown, that for $v_{k+1} = -\nabla f(x_{k+1})$, we have

$$p_{k+1} = -\nabla f(x_{k+1}) + \sum_{i=0}^{k} \frac{\nabla f(x_{k+1})^T Q p_i}{p_i^T Q p_i} p_i = -\nabla f(x_{k+1}) + \frac{\nabla f(x_{k+1})^T Q p_k}{p_k^T Q p_k} p_k \ . \tag{21}$$

Which means that for this choice, all but the last term of the sum are zero. The proof sketch for this is to show first that with this choice of v_{k+1}, we have $\nabla f(x_i)^T \nabla f(x_j) = 0$ for $j < i$. With this, we then know that for all $j < k+1$

$$0 = \nabla f(x_{k+1})^T \nabla f(x_j) = \nabla f(x_k)^T \nabla f(x_j) + \alpha_k p_k^T Q \nabla f(x_j)$$

The first term is zero, when $j < k$, from which follows that $p_k^T Q \nabla f(x_j) = 0$, when $j < k$, which allows us to set most terms in the sum of (21) to zero.

Putting everything together, we arrive at the CG algorithm, Algorithm 5. It proceeds from an initial point $x_0 = 0$, computes the initial gradient $\nabla f_0 = \nabla f(x_0)$ and sets the initial step direction to the steepest descent direction. Then, in the loop, the optimal step length α_k is computed via equation (18), the step is performed and the gradient at the new position $\nabla f_{k+1} = \nabla f(x_{k+1})$ is computed. Then, the algorithm test whether the stopping criterion is met and otherwise, the conjugate step direction is updated using equation (20).

This variation of conjugate gradients is the easiest to implement, but not the fastest implementation. A common optimisation is to pre-compute Qp_k at the beginning of the iteration, as it is reused several times later on. With this, the computation of the gradient can also be optimised as we have $\nabla f_{k+1} =$

Data: $Q \in \mathbb{R}^{n \times n}$ symmetric positive definite, $g \in \mathbb{R}^n$ $\epsilon > 0$
Result: Point x_{k+1} with $\|Qx_k + g\| \leq \epsilon$
$x_0 = 0$;
$\nabla f_0 = g$;
$p_0 = -\nabla f_0$;
for $k = 0, \ldots$ **do**
  $\alpha_k = -\frac{p_k^T \nabla f_k}{p_k^T Q p_k}$;
  $x_{k+1} = x_k + \alpha_k p_k$;
  $\nabla f_{k+1} = Q x_{k+1} + g$;
  if $\|\nabla f_{k+1}\| < \epsilon$ **then**
    stop;
  $p_{k+1} = -\nabla f_{k+1} + \frac{\nabla f_{k+1}^T Q p_k}{p_k^T Q p_k} p_k$

Algorithm 5: Conjugate Gradients

$\nabla f_k + \alpha_k Q p_k$. This brings down the cost to just a single matrix-vector multiplication per iteration and a full run of the algorithm takes at most n matrix-vector multiplications. Finally, the algorithm does only use very little additional memory, through reuse of variables, x_k, p_k and ∇f_k. This makes the algorithm also interesting for very large systems of equations, where Q is a sparse matrix with millions of rows and columns.

Note that for very large systems, the gradient update using Qp_k will accumulate numerical round-off errors over time and thus the numerically computed steps will become imprecise. For the same reason, numerical errors will cause that computed p_{k+1} will not be conjugate to $p_1, \ldots, p_k$ and this effect becomes worse as the number of steps grows. This is because later steps never get corrected to be conjugate to, for example, the very first step of the sequence. While the first problem can be solved by recomputing the gradient every once in a while, e.g., every 1000 steps, the second error is an inherent problem of the conjugate gradients method and in some cases it is beneficial to *restart* CG, by setting $p_{k+1} = -\nabla f_{k+1}$, which resets the conjugate gradient sequence back to the gradient descent. This of course should only be done if it is suspected that the round-off error could be significant.

3.3 Inexact Newton Algorithm

Finally, we can bind the results of this chapter together. The inexact Newton algorithm presented in Algorithm 6 is a direct variation of Newton's Algorithm (Algorithm 4) under the initial constraint added in the chapter, that the objective function f is convex. Otherwise, the check for $\nabla^2 f(x_k)$ to be positive definite is already more expensive than computing the inverse, thus approximating the Newton step does not save time.

In each iteration of the algorithm, we first compute η_k. Here, we used one of the choices for superlinear, but not quadratic convergence. With chosen η_k, we can compute the target stopping criterion ϵ_k and then apply it to the conjugate

Data: $x_0 \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$ with pos.definite Hessian, $0 < c_1 < 1$, $0 < \rho < 1$
Result: Estimate of global minimum x_{k+1} of f
for $k = 0, \ldots,$ *until any stopping condition is fulfilled* **do**
$\eta_k = \frac{1}{2} \min\left\{\frac{1}{2}, \sqrt{\|\nabla f(x_k)\|}\right\}$;
$\epsilon_k = \eta_k \|\nabla f(x_k)\|$;
$p_k = \texttt{conjugate\_gradient}(\nabla^2 f(x_k), \nabla f(x_k), \epsilon_k)$;
$\alpha_k = \text{backtrack}(x_k, p_k, 1.0, c_1, \rho)$;
$x_{k+1} = x_k + \alpha_k p_k$;

Algorithm 6: Newtons Algorithm

gradient algorithm (Algorithm 5) that computes an approximate Newton step. We refer to it here as `conjugate_gradient`(Q, g, ϵ), where Q is the quadratic term, g the linear offset and ϵ the solver precision. Finally, we use again a backtracking linesearch to find a suitable steplength and update the position of our algorithm. This is as always repeated until one of our standard stopping criteria is met.

4 Quasi-Newton Methods

In the previous chapters, we have worked with Newton's algorithm, a variant of which we presented in Algorithm 4. Newton's algorithm can converge quickly, but has a number of disadvantages

- Second derivatives are expensive to compute in high dimensions. On an n-dimensional optimisation problem, the Hessian has size $n \times n$. This can require a lot of memory, but also each of the entries needs to be computed. For many functions, computing the Hessian has a similar cost as n gradient computations.
- Inverting the Hessian is even more expensive.
- When the Hessian is not positive definite, the information gained from the Hessian is difficult to interpret.

These shortcomings can make it difficult to actually use Newton's method in practice. We have seen in the previous chapter that we can reduce the cost of computing the inverse somewhat by only using an approximation of the Newton step at a cost of convergence speed. In this chapter, we want to develop this idea further, by exploring the idea of a Hessian approximation. We take a look at local quadratic models

$$m_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T B_k p \ , \tag{22}$$

which are similar to the second order Taylor expansion we used to derive the Newton step, except that now we use an approximation B_k of the Hessian $\nabla^2 f(x_k)$. Using such a model, assuming B_k is positive definite, the optimal step in the model can be computed as

$$p_k = -B_k^{-1} \nabla f(x_k) \ .$$

As long as B_k is positive definite, we do not have to worry that the computed p_k is a descent direction, since when x_k is not a critical point, we have

$$\nabla f(x_k)^T p_k = -\underbrace{\nabla f(x_k)^T B_k^{-1} \nabla f(x_k)}_{>0} \ .$$

As the overall strategy of the algorithm and their performance is very similar to Newtons algorithms, these algorithms are also referred to as *Quasi Newton* methods.

If we manage to cheaply obtain an approximation $B_k \to \nabla^2 f(x^*)$ as $x_k \to x^*$, we might enjoy similar optimisation speed, while paying less of the costs. We will not obtain the same convergence, though: as we converge to the optimum quicker, our approximation errors of B_k will always stay relevant, and thus quadratic convergence is out of reach. We can however reach superlinear convergence. Looking at the list of shortcomings of Newton's method, we formulate a list of properties we would like to obtain:

- Computing B_k should not require computing any parts of the Hessian.
- The inverse, $H_k = B_k^{-1}$ should either be easy to compute, or there is a fast update equation from H_k to H_{k+1} such that $H_{k+1} = B_{k+1}^{-1}$.
- In every step, B_k (and H_k) must remain positive definite.

If we found an update equation that fulfiled all these criteria, we could simplify Newton's algorithm (Algorithm 4), as we can see in the template given by Algorithm 7. We will in the following derive the most important Quasi-Newton

Data: $x_0 \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $H_0 \in \mathbb{R}^{n \times n}$
Result: Estimate of local minimum x_{k+1} of f
for $k = 0, \ldots,$ *until any stopping condition is fulfiled* **do**
  $p_k = -H_k \nabla f(x_k)$;
  Find steplength α_k;
  $x_{k+1} = x_k + \alpha_k p_k$;
  Compute H_{k+1};

Algorithm 7: Basic Quasi-Newton Algorithm template

method, BFGS, named after its discoverers, Broyden, Fletcher, Goldfarb, and Shanno. The BFGS algorithm is one of the most used optimisation algorithms and some variant of it is included in almost every optimisation toolbox for medium-sized, unconstrained optimisation problems. We will not derive it using the approach that was historically used to arrive at this algorithm, but instead we will introduce its core ideas, piece by piece.

4.1 The Secant Equation

As a running example to show the effects of the core ideas behind BFGS, let us assume for now that the function to optimise is a simple quadratic function

$$f(x) = g^T x + \frac{1}{2} x^T Q x \ ,$$

where g and Q are unknown and the function has derivatives:

$$\nabla f(x) = Qx + g$$
$$\nabla^2 f(x) = Q$$

Let us pretend, that we are at the end of the kth iteration of Algorithm 7. We used an existing approximation B_k to create a local model m_k around the point x_k using equation (22) and arrived at a new iterate $x_{k+1} = x_k + \alpha_k p_k$. At this new point, we can compute a new derivative $\nabla f(x_{k+1})$. We now have to propose a new model m_{k+1}. As for the previous models introduced, we require that

$$m_{k+1}(0) = f(x_{k+1})$$
$$\nabla m_{k+1}(0) = \nabla f(x_{k+1}) \ .$$

Which is fulfiled by the model (22). Additionally, if it describes the function well, then we would at least expect that it predicts the value of the gradient at the previous point

$$\nabla m_{k+1}(\underbrace{x_k - x_{k+1}}_{-\alpha_k p_k}) = \nabla f(x_k) \ .$$

Using the definition m_k, we can derive

$$\begin{aligned}
&\nabla m_{k+1}(x_k - x_{k+1}) = \nabla f(x_k) \\
\Leftrightarrow &\nabla f(x_{k+1}) + B_{k+1}(x_k - x_{k+1}) = \nabla f(x_k) \\
\Leftrightarrow &\quad B_{k+1}(x_k - x_{k+1}) = \nabla f(x_k) - \nabla f(x_{k+1}) \\
\Leftrightarrow &\quad B_{k+1}(x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k) \qquad (23) \\
\Leftrightarrow &\quad B_{k+1}(x_{k+1} - x_k) = Q(x_{k+1} - x_k) \ .
\end{aligned}$$

The last line only holds for our specific choice of f as a quadratic function, but it indicates that in this case, if our model predicts correctly the derivatives, then its quadratic terms must be the same.

For general f, it is equation (23) that is of interest, even though in the general case, it is only an approximation: if we set $B_k = \nabla^2 f(x_k)$ this relation will not hold for most functions due to higher order terms. However, it is still approximately true close to the optimum: as steps become shorter, the higher order terms become less important. We can phrase this requirement more succinctly, if we define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ and can also derive a variant for the inverse $H_{k+1} = B_{k+1}^{-1}$:

$$B_{k+1} s_k = y_k \qquad (24)$$

$$H_{k+1} y_k = s_k \qquad (25)$$

Both equations are referred to as the *Secant Equation* and the second version can be derived from the first version by multiplying H_{k+1} from the left.

Unfortunately, for a given choice of y_k and s_k, there might not exist a B_{k+1} that fulfils the secant equation and is positive definite. To show this, we multiply (24) with s_k^T from the left and obtain

$$s_k^T B_{k+1} s_k = s_k^T y_k \ . \qquad (26)$$

If $s_k^T y_k < 0$ there does not exist a positive definite update, as $s_k^T B_{k+1} s_k < 0$ implies the existence of a negative eigenvalue in B_{k+1}. Thus, for some iterates x_{k+1} it is impossible to use the secant equation and keep the matrix positive definite. For now, we will assume that $s_k^T y_k > 0$ and will discuss ways to ensure this later on.

4.2 Updating the Hessian Estimate using the Secant equation

There are infinitely many ways to pick a new matrix B_{k+1} that can fulfil the secant equation (24). Our next idea is that we include some of our requirements:

our goal is to find a cheap rule that adapts B_k (or H_k) . One of the cheapest approaches is to use a rank-1 update:

$$B_{k+1} = B_k + \gamma v v^T,\ \gamma \in \mathbb{R},\ v \in \mathbb{R}^n \ .$$

We will next set out to find the value of v that fulfils the secant equation.

Lemma 5. *The unique update formula of the form*

$$B_{k+1} = B_k + \gamma v v^T,\ v \in \mathbb{R}^n \ .$$

that fulfils the secant equation is

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \ . \tag{27}$$

and the inverse H_{k+1} fulfils

$$H_{k+1} = B_{k+1}^{-1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k} \ . \tag{28}$$

Before we proof this statement, let us get an intuition about it. Remember that y_k was the observed difference of gradients and s_k the step of the last iteration. Then, $y_k - B_k s_k$ is the difference between the observed difference of gradients and the predicted difference by the approximation of the Hessian B_k. The update equation then changes B_k to correct for this error in prediction.

Proof. We begin by inserting our update equation in (24)

$$\begin{aligned} & B_{k+1} s_k && = y_k \\ \Leftrightarrow & \left(B_k + \gamma v v^T\right) s_k && = y_k \\ \Leftrightarrow & B_k s_k + \gamma v v^T s_k && = y_k \\ \Leftrightarrow & \gamma (v^T s_k) v && = y_k - B_k s_k \end{aligned}$$

The term $\gamma(v^T s_k)$ is a scalar, while on the right hand side, the quantity $y_k - B_k s_k$ is a vector in $\mathbb{R}^n$. the only way both sides can be equal is if $v \propto y_k - B_k s_k$. Since γ can be any value, we can start by setting $v = y_k - B_k s_k$ and find the correct factor using γ. Assuming that at least one element of v is not zero, the equation can only be fulfilled if $\gamma(v^T s_k) = 1$ and thus

$$\gamma = \frac{1}{v^T s_k} = \frac{1}{(y_k - B_k s_k)^T s_k} \ .$$

The derivation of H_{k+1} follows in a similar fashion starting from (25). It can be verified that it holds $H_{k+1} = B_{k+1}^{-1}$. □

This equation already fulfils most of our requirements: We found an update equation of H_{k+1} from H_k that does not require any Hessian information and is easy to compute. Moreover, one can show, that under the assumption that the denominator in (28) is never zero, this update will learn the true Hessian on a strongly convex quadratic function in n steps.

Theorem 7. *Let $f(x) = g^T x + \frac{1}{2} x^T Q x$, $x \in \mathbb{R}^n$, where Q is symmetric positive definite. Consider the sequence of iterates $H_1, H_2, \ldots$ generated by Algorithm (7) starting from any positive definite H_0 using the SR1 update equation (28). Assume further, that in each iteration $(s_k - H_k y_k)^T y_k \neq 0$, then $H_n = Q^{-1}$.*

Proof. The proof idea is as follows: we will first show that H_{k+1} fulfils the secant equation not only for the current pair s_k, y_k, but also for all previous pairs. Then, we will use a relation between s_k and y_k on quadratic functions to create a system of equations using all s_k, y_k pairs that shows that $H_n = Q^{-1}$ under the condition that s_k are independent. We finally show that under the assumption $(s_k - H_k y_k)^T y_k \neq 0$ they must be independent.

We will show first by induction, that H_{k+1} fulfils the secant equations for all iterations, that is,

$$H_{k+1} y_j = s_j, \; j = 1, \ldots, k \; . \tag{29}$$

In the first iteration, H_1 fulfils this requirement automatically due to construction with the secant equation. Let us now assume that for some $k > 0$, H_k fulfils assumption (29). We will now show that it also holds for H_{k+1}.
We will need the following result that holds on quadratic functions:

$$\begin{aligned} s_k^T y_j &= (x_{k+1} - x_k)^T (\nabla f(x_{j+1}) - \nabla f(x_j)) \\ &= (x_{k+1} - x_k)^T (Q x_{j+1} - Q x_j) \\ &= (Q x_{k+1} - Q x_k)^T (x_{j+1} - x_j) \\ &= y_k^T s_j \end{aligned}$$

From which also follows in more general terms that $y_j = Q s_j$.
For any $j < k$, we have due to our induction assumption and our previous result, that

$$\begin{aligned} (s_k - H_k^T y_k)^T y_j &= s_k^T y_j - y_k^T \underbrace{H_k y_j}_{s_j} \\ &= s_k^T y_j - y_k^T s_j = s_k^T y_j - s_k^T y_j = 0 \end{aligned}$$

Inserting this result into (28), we obtain

$$H_{k+1}y_j = \overbrace{H_k y_j}^{s_j} + \frac{(s_k - H_k y_k)\overbrace{(s_k - H_k y_k)^T y_j}^{0}}{(s_k - H_k y_k)^T y_k} = s_j$$

As this holds for any $j < k$ and the denominator is not zero by assumption, we have shown that (29) holds for H_{k+1}, which completes the proof by induction.
We have now shown that for our choice of function holds

$$s_j = H_{k+1}y_j = H_{k+1}Qs_j,\ j = 0, \dots, k,\ k = 0, \dots, n-1$$

In iteration $k = n-1$, there are n vectors $s_0, \dots, s_{n-1}$ that are mapped to themselves by the matrix $H_{k+1}Q$. If these n vectors are independent, then they form a basis and $H_{k+1} = H_n = Q^{-1}$.
For contradiction, assume now, that in iteration k, s_k is linearly dependent on $s_0, \dots, s_{k-1}$, that means there exists a linear combination $s_k = \sum_{j=0}^{k-1} w_j s_j$. Then, using $y_j = Qs_j$, we have for our quadratic function that $y_k = Qs_k = \sum_{j=0}^{k-1} w_j y_j$ and thus for the denominator holds

$$\begin{aligned}
(s_k - H_k y_k)^T y_k &= \left(s_k - H_k \left(\sum_{j=0}^{k-1} w_j y_j \right) \right)^T y_k \\
&= \left(s_k - \sum_{j=0}^{k-1} w_j H_k y_j \right)^T y_k \\
&= \left(s_k - \sum_{j=0}^{k-1} w_j s_j \right)^T y_k \\
&= (s_k - s_k)^T y_k = 0 \ .
\end{aligned}$$

This contradicts our assumption that the denominators are not zero, and thus all s_k must be linearly independent. □

The previous result can be strengthened insofar as one can show that on a strictly convex quadratic function, Algorithm 7 using the SR1 update and fixed $\alpha_k = 1$ reaches the optimum after n iterations. Moreover, under a small condition on the absolute value of the denominator, and assuming that $x_k \to x^*$, one can show convergence $B_k \to \nabla^2 f(x^*)$.

Unfortunately, even if $s_k^T y_k > 0$, the update equations (27)&(28) can sometimes lead to indefinite matrices. To see this, we take s_k and compute its

quadratic form with B_{k+1}

$$\begin{aligned} x^T B_{k+1} x &= x^T B_k x + x^T \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} x \\ &= x^T B_k x + \frac{x^T (y_k - B_k s_k)(y_k - B_k s_k)^T x}{(y_k - B_k s_k)^T s_k} \\ &= x^T B_k x + \frac{\left(x^T (y_k - B_k s_k)\right)^2}{(y_k - B_k s_k)^T s_k} \end{aligned}$$

While the first term is positive provided that B_k is positive definite, the sign of the second term depends on $(y_k - B_k s_k)^T s_k = y_k^T s_k - s_k^T B_k s_k$. We assumed in the beginning that $y_k^T s_k > 0$. However, we subtract $s_k^T B_k s_k > 0$ and thus, when this term is large enough, the sign of the whole term can become negative and one can construct explicit examples that result in indefinite B_k.

The SR1 equation (27) alone does not lead to a positive definite update. And from the derivation we see, that we used all our degrees of freedom and there are no parameters left to improve this. Thus, the simple rank-1 update is not powerful enough to ensure positive definiteness. We will therefore investigate the next more powerful update, a rank-2 update:

Lemma 6. *Assume B_k is positive definite and $s_k^T y_k > 0$. Then the update*

$$B_{k+1} = B'_{k+1} + \frac{y_k y_k^T}{y_k^T s_k} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \quad . \tag{30}$$

fulfils the secant equation (24) and is positive definite. Further, if $H_k = B_k^{-1}$, then

$$H_{k+1} = B_{k+1}^{-1} = H_k + \frac{s_k^T y_k + y_k^T H_k y_k}{(s_k^T y_k)^2} s_k s_k^T - \frac{H_k y_k s_k^T + s_k y_k^T H_k}{s_k^T y_k} \quad . \tag{31}$$

Proof. Consider the update

$$B_{k+1} = \underbrace{B_k + \beta u u^T}_{B'_{k+1}} + \gamma v v^T, \; u, v \in \mathbb{R}^n \quad .$$

Since the SR1 update we just derived holds for any choice of B_k, we can replace B_k in equation (27) by B'_{k+1} and obtain

$$B_{k+1} = B'_{k+1} + \frac{(y_k - B'_{k+1} s_k)(y_k - B'_{k+1} s_k)^T}{(y_k - B'_{k+1} s_k)^T s_k} \quad . \tag{32}$$

Thus, no matter how we choose β and u we can always find values for γ and v that fulfil the secant equation. Inserting the definition of B'_{k+1} leads to the denominator

$$(y_k - B'_{k+1} s_k)^T s_k = y_k^T s_k - \left(B_k s_k + \beta u u^T s_k\right) s_k \quad .$$

Next, we choose u such, that the term in parenthesis is zero, and thus when $y_k^T s_k > 0$, the SR1 update is positive definite. Using a similar argument as before, we arrive at $u = B_k s_k$ and

$$\beta = -\frac{1}{u^T s_k} = -\frac{1}{s_k^T B_k s_k} \ .$$

With this choice, we have $B'_{k+1} s_k = 0$ and thus, when we insert our result into (32) we obtain (30).
To show positive definiteness, assume that B_k is positive definite. Due to this, and since B_k is symmetric, there exists matrix A such, that $B_k = A^T A$ and $B_k^{-1} = A^{-1} A^{T^{-1}}$. Pick an arbitrary $x \in \mathbb{R}^d \neq 0$. Further, let $\tilde{x} = Ax$, $\tilde{y} = Ay_k$ and $\tilde{s} = As_k$. Then

$$\begin{aligned} x^T B_{k+1} x &= x^T B_k x - \frac{x^T B_k s_k s_k^T B_k x}{s_k^T B_k s_k} + \frac{x^T y_k y_k^T x}{y_k^T s_k} \\ &= \tilde{x}^T \tilde{x} - \frac{\tilde{x}^T \tilde{s} \tilde{s}^T \tilde{x}}{\tilde{s}^T \tilde{s}} + \frac{\tilde{x}^T B_k^{-1} \tilde{y} \tilde{y}^T B_k^{-1} \tilde{x}}{\tilde{y}^T B_k^{-1} \tilde{s}} \\ &= \tilde{x}^T \left(I_n - \frac{\tilde{s}\tilde{s}^T}{\tilde{s}^T \tilde{s}} \right) \tilde{x} + \frac{(\tilde{x}^T B_k^{-1} \tilde{y})^2}{\tilde{y}^T B_k^{-1} \tilde{s}} \end{aligned}$$

It can be verified, that in the first term, $I_n - \frac{\tilde{s}\tilde{s}^T}{\tilde{s}^T \tilde{s}}$ is a positive definite matrix, with all eigenvalues 1, except one eigenvalue that is 0 with eigenvector $\tilde{s}$. As the last term is non-negative we have that $x^T B_{k+1} x > 0$ for all directions that are not parallel to s_k. Finally, take $x = s_k$ and thus, $\tilde{x} = \tilde{s}$ and we obtain

$$\begin{aligned} s_k^T B_{k+1} s_k &= \tilde{s}^T \left(I_n - \frac{\tilde{s}\tilde{s}^T}{\tilde{s}^T \tilde{s}} \right) \tilde{s} + \frac{(\tilde{s}^T B_k^{-1} \tilde{y})^2}{\tilde{y}^T B_k^{-1} \tilde{s}} \\ &= \frac{(\tilde{s} B_k^{-1} \tilde{y})^2}{\tilde{y}^T B_k^{-1} \tilde{s}} = \frac{(s_k^T y_k)^2}{y_k^T s_k} = s_k^T y_k > 0 \end{aligned}$$

where the last part holds by assumption. Finally, we can verify that H_{k+1} is the inverse of B_{k+1}. □

This update is called the BFGS update and fulfils all three of our requirements. However, due to our choice of additional direction vector, the convergence theory of BFGS is far less straight-forward as it was for SR1, even though convergence to the true Hessian can be shown. Moreover, the final equation is not quite satisfying. During its derivation, we applied the rank-2 update to B_k and then inverted the result. The two resulting equations are not as nicely symmetric as in the SR1 update, and if we change the roles of both matrices by applying the rank-2 update to H_k and then invert the result, we do not obtain our derived update equation of B_k. Indeed, this second derivation is known as

DFP, named after its discoverers, Davidon, Fletcher, Powell. The DFP formula has similar theoretical properties as the BFGS update, but empirically performs worse.

4.3 A Linesearch with strict Wolfe conditions

So far, all our updates only work under the assumption that $y_k^T s_k > 0$. We are investigating this requirement next. Remembering, that $s_k = x_{k+1} - x_k = \alpha_k p_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$, we can reformulate this requirement to

$$\begin{aligned} & y_k^T s_k > 0 \\ \Leftrightarrow & (\nabla f(x_{k+1}) - \nabla f(x_k))^T (\alpha_k p_k) > 0 \\ \Leftrightarrow & \quad \nabla f(x_{k+1})^T p_k - \nabla f(x_k)^T p_k > 0 \\ \Leftrightarrow & \quad p_k^T \nabla f(x_k + \alpha p_k) > p_k^T \nabla f(x_k) \end{aligned}$$

This, is very similar to the curvature condition of the Wolfe Conditions, Definition 3, equation (6) that we restate here:

$$p_k^T \nabla f(x_k + \alpha p_k) \geq c_2 p_k^T \nabla f(x_k), \; c_2 \in (0, 1)$$

Comparing the inequalities carefully and using that $p_k^T \nabla f(x_k) < 0$, we can see that if the curvature condition holds, then $s_k^T y_k > 0$. Thus, if we choose our steplength α_k such, that the Wolfe conditions are fulfilled, then our BFGS update (30) exists. Since we already go through the trouble of ensuring the Wolfe conditions, we can go one step further and fulfil the strict Wolfe conditions. This is the outcome of the next algorithm. Before we introduce it, we are going to introduce its core principles. The algorithm keeps track of a steplength interval (l, u) that contains a steplength α^* that fulfils the Wolfe conditions. At each iteration, it is updated to fulfil the following conditions

1. There exists an interval (l, u'),$u' \leq u$ that fulfils the sufficient decrease condition
2. $\nabla f(x_k + l \cdot p_k)^T p_k < 0$
3. Either
   (a) $x_k + u \cdot p_k$ does not fulfil the sufficient decrease condition
   (b) $\nabla f(x_k + u \cdot p_k) > 0$
   (c) $f(x_k + l \cdot p_k) < f(x_k + u \cdot p_k)$.

Let us try to understand these conditions. From the proof of Lemma 2 we know that if we find an u that does not fulfil the sufficient decrease condition (condition 3.a), then there exists an $u' < u$ so that all points on the interval $(0, u')$ fulfil the sufficient decrease condition. Further, we know that if $0 < c_1 < c_2 < 1$, there exists a point within that interval that also fulfils the strong curvature condition. This means that, once we pick $l > 0$, we have to ensure

Data: $x, p \in \mathbb{R}^n$, $0 < c_1 < c_2 < 1$, $\alpha_{\text{init}} > 0$
Result: Steplength α^* that fulfils the strong Wolfe conditions
Let $g(\alpha) = f(x_k + \alpha p_k)$;
$l \leftarrow 0$, $u \leftarrow \alpha_{\text{init}}$;
while *True* **do**
    if $g(u) > g(0) + c_1 \cdot u \cdot g'(0)$ *or* $g(u) > g(l)$ **then**
        break;
    if $|g'(u)| < c_2|g'(0)|$ **then**
        $\alpha^* = u$;
        stop;
    if $g'(u) > 0$ **then**
        break;
    else
        $u \leftarrow u \cdot 2$;
while *True* **do**
    $a = (l + u)/2$;
    if $g(a) > g(0) + c_1 \cdot a \cdot g'(0)$ *or* $g(a) > g(l)$ **then**
        $u \leftarrow a$;
    else
        if $|g'(a)| < c_2|g'(0)|$ **then**
            $\alpha^* = a$;
            stop;
        if $g'(a) < 0$ **then**
            $l \leftarrow a$;
        else
            $u \leftarrow a$;

Algorithm 8: Strict Wolfe condition Linesearch

that when increasing l we do not accidentally exclude the points that fulfil the strong curvature condition. Further, once we shrink the interval such, that $u < u'$, we have to find different conditions that ensure that we do not exclude the interval of interest.

For the remaining conditions, let us define $g(\alpha) = f(x_k + \alpha p_k)$. Condition 2 is equivalent to $g'(l) < 0$, that is, the function values decrease when slightly increasing l. Since l does not fulfil the strong Wolfe conditions (otherwise the algorithm would terminate), it holds $g'(l) < c_2 g'(0) < c_1 g'(0)$ and thus small increases to l will reduce function values by more than the sufficient decrease condition requires which can be used to show that there is a point in (l, u) that fulfils the curvature condition.

Finally, when u fulfils the sufficient decrease condition, then conditions 3b) and 3c) ensure that there is a minimum of g in the interval (l, u). There might be multiple critical points, but one of them must be a local minimum. A local minimum at position $\alpha^* \in (l, u)$ always fulfils the curvature condition and since it is a minimum, it must hold $g(\alpha^*) < g(u) \leq g(0) + uc_1 \leq g(0) + \alpha^* c_1 g'(0)$ and thus the point fulfils the sufficient decrease condition. Condition 3.2 ensures the existence of the minimum via the intermediate value theorem (see Appendix, Theorem 10): since condition 2 requires $g'(l) < 0$ and 3.2 requires $g'(u) > 0$, there must be an $\alpha^* \in (l, u)$ such, that $g'(\alpha^*) = 0$. Similarly, the third condition requires that $g(u) > g(l)$ and since $g'(u) < 0$, the function must contain a minimum on the interval (l, u) due to the mean-value theorem (see Appendix, Theorem 11).

Algorithm 8 follows these ideas. It is partitioned in two parts, represented by two while loops. It first initialises $l = 0$, which fulfils conditions 1 and 2. The first loop then tries to find any u that fulfils condition 3. It does that by geometrically increasing u until one of the conditions 3a-3c are fulfilled. If any point fulfils the strong Wolfe conditions, the algorithm terminates with that point as answer. If any u is found that fulfils condition 3, the first while-loop stops.

The second while loop then tries to find a sufficient steplength within the interval. In each iteration, the midpoint a is evaluated. If the point fulfils conditions 1 or 2, l is set to a. Otherwise, if a fulfils condition 3, u is updated, and due to the way the conditions are set up, each point must fulfil at least one of those requirements, or fulfil the strong curvature conditions, in which case the algorithm stops.

4.4 The BFGS algorithm

Data: $x_0 \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $0 < c_1 < c_2 < 1$
Result: Estimate of local minimum x_{k+1} of f
$H_0 = I_n$ **for** $k = 0, \ldots,$ *until any stopping condition is fulfilled* **do**
  $p_k = -H_k \nabla f(x_k)$;
  $\alpha_k = \texttt{wolfe\_search}(x_k, p_k, 1.0, c_1, c_2)$;
  $x_{k+1} = x_k + \alpha_k p_k$;
  $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$;
  $s_k = \alpha_k p_k$;
  $H_{k+1} = H_k + \frac{s_k^T y_k + y_k^T H_k y_k}{(s_k^T y_k)^2} s_k s_k^T - \frac{H_k y_k s_k^T + s_k y_k^T H_k}{s_k^T y_k}$;

Algorithm 9: BFGS Algorithm

We can finally put together all the parts. We can turn our Quasi-Newton algorithm template, Algorithm 7 into the real BFGS algorithm, Algorithm 9, by inserting the BFGS update equations of H_k, equation (31) and use the algorithm that fulfils the strict Wolfe condition, Algorithm 8 for finding the steplength. In the algorithm, we call this `wolfe_search` with arguments x_k, p_k, α_{init}, c_1 and c_2.

5 Constrained Continuous optimisation

We will now look at minimisation problems that are constrained, that means we try to minimise our objective function $f : \mathbb{R}^n \to \mathbb{R}$ under a set of additional constraints that the solution must fulfil. More formally, we have

$$\begin{aligned} &\min_x f(x) \\ &\text{s.t. } c_i(x) = 0,\ i \in \mathcal{E} \\ &\wedge\ c_i(x) \leq 0,\ i \in \mathcal{I} \end{aligned} \tag{33}$$

Here, $c_i : \mathbb{R}^n \to \mathbb{R}$,$i = 1, \ldots, m$ are constraint functions, which are partitioned in equality and inequality constraints. The equality constraints have $c_i(x) = 0$, and their indices are included in the set $\mathcal{E}$. The inequality constraints have $c_i(x) \leq 0$ and their indices are included in $\mathcal{I}$. We will assume that the functions c_i are continuously differentiable.

We call a solution feasible, if it fulfils all constraints. The set

$$\mathcal{C} = \{x \in \mathbb{R}^n \mid c_i(x) = 0, c_j(x) \leq 0, \forall i \in \mathcal{E}, \forall j \in \mathcal{I}\}$$

is called the *feasible set*. Using the feasible set, we can write (33) shorter as

$$\min_{x \in \mathcal{C}} f(x)\ .$$

Similar to unconstrained problems, we need to define local optima and conditions for local optima. However, due to the inclusion of constraints, the mathematical complexity of the theorems increases and our usual necessary and sufficient conditions do not necessarily apply. To see this, take a look at the following example

Examples: Let $f(x) = x$, $x \in \mathbb{R}$ and the feasible set $\mathcal{C} = [0, 1]$. The minimum point in this set can be guessed easily as $x = 0$. However, we have $\nabla f(x) = 1$ for all $x \in \mathbb{R}$ and thus the necessary conditions for unconstrained critical points, Theorem 1 do not hold at $x = 0$.

In this example, another problem becomes apparent: since the minimum $x^* = 0$ is exactly on the boundary of the set, there exists no open neighbourhood $\mathcal{N}$ around it, as was required by the definition of (weak) local minima in the unconstrained case, Definition 1. Allowing closed neighbourhoods does not work either, since in the example above any $x \in [0, 1]$ is minimiser of a closed set, $\mathcal{N} = [x, 1]$. Still, if the optimum does not lie on the boundary (for example, when we instead optimised $f(x) = (x - \frac{1}{2})^2$, using the open neighbourhood approach still works, as we can always make the open set small enough to fit inside $\mathcal{C}$.

Taking all of this into account, we will adapt our definition that we still require an open neighbourhood on $\mathbb{R}^n$ that includes x^*, and then intersect this set with $\mathcal{C}$ to exclude all points that are not feasible.

Definition 4 (Constrained Minimisers)**.** *When solving the constrained problem* (33)*, a point* x^* *is a*

1. *weak local minimiser of* f *if there exists a neighbourhood* $\mathcal{N} \subseteq \mathbb{R}^n$ *around* x^* *with*
$$f(x^*) \leq f(x),\ \forall x \in \mathcal{N} \cap \mathcal{C}$$
2. *strict/strong local minimiser of* f *if there exists a neighbourhood* $\mathcal{N} \subseteq \mathbb{R}^n$ *around* x^* *with*
$$f(x^*) < f(x),\ \forall x \in \mathcal{N} \setminus \{x^*\} \cap \mathcal{C}$$
3. *global minimiser of* f *if*
$$f(x^*) \leq f(x)\ \forall x \in \mathcal{C}$$

Constraints impact our algorithms more than just the basic definition of a minimum. They also often complicate finding a critical point. We have already seen in our initial example, that the gradient is not necessarily zero at an optimum and thus the sufficient first-order conditions for unconstrained optima do not hold. Algorithmically, constraints also complicate our search for an optimum as they limit our ability to move through the space: if we require our algorithm to only propose x_k that fulfil the constraints, then we cannot use a search direction that immediately leads us outside the feasible area. This leads to the question how we can formalise that a constraint prevents a step direction from creating a feasible point. We will start by building some intuition using the example of a single linear constraint.

Example: As first example, consider some function f with additional inequality constraint $c_1(x) = a^T x + b \leq 0$ with $a = (1, -1)^T$ and $b = 0$. We pick $x = (0, 1)^T$. It fulfils $c_1(x) = a^T x + b = -1 < 0$, thus x is feasible. Let us take any step direction $p \in \mathbb{R}^2$. A step then involves picking a point along the line $x + \alpha p$, $\alpha > 0$. Inserting this into the constraint, we have

$$a^T(x + \alpha p) + b = a^T x + b + \alpha a^T p = -1 + \alpha a^T p \leq 0$$

If we have $a^T p \leq 0$, then $x + \alpha p$ will be feasible for any $\alpha > 0$. If $a^T p > 0$ then for $\alpha \leq 1/a^T p$, we have $a^T(x + \alpha p) + b \leq 0$, while larger steps will violate the constraint. Thus, in each direction, we find possible step lengths and thus no direction is excluded by the constraint.

As second example, we now pick $x = (1, 1)^T$, for which $a^T x + b = 0$. In this case, not all p are possible. If we pick any p such, that $a^T p > 0$, we have

$$a^T(x + \alpha p) + b = \alpha a^T p \geq 0\ ,$$

which means that for any $\alpha > 0$ we have that $x + \alpha p$ is not feasible and

thus any direction p with $p^T a > 0$ is excluded by the constraint.

Finally, let us change the inequality constraint to an equality constraint $a^T x + b = 0$ and pick the feasible $x = (1, 1)^T$. With this, we have

$$a^T(x + \alpha p) + b = \alpha a^T p = 0 \ ,$$

and thus, a direction can only create a step that leads to a feasible point if $a^T p = 0$

This example shows that for inequality constraints it makes a difference whether we fulfil them with equality or not. As long as they are not fulfilled with equality, we can ignore them while searching for a possible step direction. Indeed, this holds in general as long as the constraint is continuous: if a point fulfils $c_1(x) < 0$ there will be an open neighbourhood $\mathcal{N}$ around x with $c_i(y) < 0, \forall y \in \mathcal{N}$.

Unfortunately, our approach of constructing lines to show that a direction leads to feasible points is flawed in the case of non-linear constraints. While for linear constraints it was possible for us to construct a ray starting at x with direction p, and showed that there exist possible steplengths in that direction, this approach does not work in general. For nonlinear constraints a line might never create a feasible point, as we show in the next example:

Example: Consider the equality constraint $c_1(x) = x^T x - 1 = 0$ for $x \in \mathbb{R}^2$. Let us create a Taylor expansion of c_1 around the feasible point $x = (1, 0)$.

$$c_1(x + \alpha p) = c_1(x) + \alpha \nabla c_1(x)^T p + R(\alpha p) = 2\alpha x^T p + R(\alpha p) \ .$$

Since the remainder $\lim_{\alpha \to 0} R(\alpha p)/\alpha \to 0$, for small α, the term $2\alpha x^T p$ will dominate $R(\alpha p)$ and thus, if we pick $x^T p \neq 0$, the constraint will be violated for all $\alpha > 0$ small enough[a]. Thus, we have to choose p such that $x^T p = 0$, for example $p = (0, 1)$. However, $x + \alpha p = (1, \alpha)$ has $c_1(x) = 1 + \alpha^2 - 1 = \alpha^2$, which does not fulfil the constraint, unless $\alpha = 0$.

[a] Since the feasible area is a circle, it might be that in the chosen direction we might cross the circle later again, but all points in-between are not feasible.

This example showed two things: for a direction to be feasible on an equality constraint, we need $\nabla c_i(x)^T p = 0$, as otherwise we can not stay on the constraint. If we think of $\nabla c_i(x)$ as normal of the constraint line $c_i(x) = 0$ we have to perform a step orthogonal to the normal. However, this is not a sufficient condition as we also have showed that linear paths in that direction might not contain any feasible points. Intuitively following the idea of the nor-

mal of the constraint line, if the constrained line is curved, then the normal will change as soon as we even make a tiny step along the constraint. We will thus generalise our idea of a descent direction to include curved paths:

Definition 5. *Let $\mathcal{C} \subset \mathbb{R}^n$ be the feasible set. We call p a tangent of $\mathcal{C}$ at $x \in \mathcal{C}$ if there exists an $u > 0$ and a function $z(\alpha) : [0, u) \to \mathcal{C}$ differentiable at $\alpha = 0$, with $z(0) = x$ and $z'(0) = p$*

We call the set of p that fulfils this condition the tangent cone at x, $T_{\mathcal{C}}(x)$.

What this definition says is a generalisation of our first example. In the first example, we constructed paths of the form $z(\alpha) = x + \alpha p$ that locally created feasible points in direction p. Here, we now allow paths that are curved as well. Their derivative $\nabla z(\alpha)$ at $\alpha = 0$ is then the vector we include as tangent. Going back to our example, we have

Example: For the constraint $c_1(x) = x^T x - 1$, pick a feasible $x = (1, 0)$. We can construct the path

$$z(\alpha) = \begin{pmatrix} \sqrt{1 - \alpha^2} \\ \alpha \end{pmatrix}$$

For $\alpha < 1 = u$, we have $c_1(z(\alpha)) = 1 - \alpha^2 + \alpha^2 - 1 = 0$ and $\nabla z(0) = (0, 1)^T$. Thus, $(0, 1)^T$ is an element of $T_{\mathcal{C}}(x)$.

This definition of the tangent cone $T_{\mathcal{C}}(x)$ is a purely geometric one. We only used the set of feasible points $\mathcal{C}$ to create paths $z(\alpha)$ that eventually end up at x. The constraints limit from which direction such a path can reach x but we do not use their algebraic formulation. However, this definition already allows us to proof first order necessary conditions for the critical points of a constrained problem.

Theorem 8. *If $x^* \in \mathcal{C}$ is a local minimiser of f, then*

$$\nabla f(x^*)^T p \geq 0, \ \forall p \in T_{\mathcal{C}}(x^*).$$

Proof. For contradiction, assume there exists $p \in T_{\mathcal{C}}(x^*)$ with $\nabla f(x^*)^T p < 0$. Then there exists a $u > 0$ and a function $z : [0, u) \to \mathcal{C}$ with $\nabla z(\alpha) = p$. We now take a look at the function $g(\alpha) = f(z(\alpha)) - f(x^*)$. We have

$$g'(\alpha) = \nabla f(x^*)^T p < 0 \ .$$

The rest of the proof now follows exactly the proof of Theorem 1. Following the same line of arguments, we can show that there exists $\alpha' \leq u$ such, that $g(\alpha) < g(0)$ for all $0 < \alpha < \alpha'$. This contradicts that x^* is a local optimum. □

It is worth to take a moment to compare Theorem 8 to the unconstrained first order conditions, Theorem 1. While previously $\nabla f(x^*) = 0$ was required, we now require a somewhat weaker condition depending on our constraints. if an unconstrained optimum, that is a point that fulfils $\nabla f(x^*) = 0$ is feasible, then this point always fulfils Theorem 8, as $0 \in T_{\mathcal{C}}(x)$, $\forall x \in \mathcal{C}$. Otherwise, our new theorem also allows us to stop if there is no tangent vector p that is a descent direction (that is, $\nabla f(x^*)^T p < 0$) in which a step can fulfil the constraints. In such a situation, we are exactly on the constraint and the gradient of the function points exactly away from the constraint normal.

Unfortunately, this theorem is difficult for us to check. As the tangent cone $T_{\mathcal{C}}(x)$ is a pure geometric construct, it is not directly apparent, how it is connected to our algebraic description of our constraints. However, in some cases, it is easy to connect both.

Lemma 7. *Let us consider a n-dimensional optimisation problem with m linear equality constraints, $c_i(x) = a_i^T x + b = 0$, $i = 1, \dots, m$, $\|a\| > 0$ and no inequality constraints. Then the tangent cone is given by*

$$T_{\mathcal{C}}(x) = \{p \in \mathbb{R}^n \mid a_i^T p = 0\} \ .$$

The proof of this lemma is left as an exercise. In the next chapter, we will generalise this problem, but for the remainder of this chapter, we will consider only optimisation of problems with linear equality constraints.

5.1 Minimisation with linear equality constraints

For this section, we consider minimisation problems with m linear equality constraints

$$\begin{aligned} &\min_x f(x) \\ &\text{s.t. } a_i^T x + b_i = 0, \ i \in \{1, \dots, m\} \end{aligned} \tag{34}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $a_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$. We will write a shorthand for it by defining $A \in \mathbb{R}^{m \times p}$ as a matrix where the ith row is a_i^T and additionally introduce a vector $b \in \mathbb{R}^n$ with elements b_i. With this, we have

$$\begin{aligned} &\min_x f(x) \\ &\text{s.t. } Ax + b = 0 \ . \end{aligned} \tag{35}$$

These problems are quite common in constrained optimisation and they are simple enough to serve as an introductory example of constraint problems. We have seen in Lemma 7 that for these problems, the tangent cone is given by

$$T_{\mathcal{C}}(x) = \{p \in \mathbb{R}^n \mid Ap = 0\}$$

and by Theorem 8 we know that a critical point $x^* \in \mathcal{C}$ has $-\nabla f(x^*)^T p$ for all $p \in T_{\mathcal{C}}(x^*)$. What is missing is a way to formalise this test. To create such

a test, we need Farkas Lemma, a theorem of the alternative. It connects the geometric object of a closed convex cone to a separating hyperplane: either, a point is inside the cone, or we find a hyperplane that separates it from it. It is the key result underpinning most of constrained optimisation theory. We will define a cone in the following way:

Definition 6 (Cone)**.** *Let $K \in \mathbb{R}^n$. We call K a cone if*

$$x \in K \Rightarrow \alpha x \in K,\ \alpha > 0.$$

We call K pointed, if

$$x \in K \text{ and } -x \in K \Rightarrow x = 0 \ ,$$

and K is called convex, if

$$x, y \in K \Rightarrow x + y \in K \ .$$

Finally, we call K a closed cone, if K is a closed set. Especially this means that $0 \in K$.

An example of a closed convex pointed cone is the set $K = \{p \in \mathbb{R}^n \mid Ap \leq 0\}$, where $A \in \mathbb{R}^{m \times n}$, and $Ap \leq 0$ is a shorthand for $A_i^T p \leq 0$ for $i = 1, \ldots, m$. A real world example for such a cone is an ice cone. In our case, the tangent cone $T_{\mathcal{C}}(x)$ given in Lemma 7 fulfils the definition of a closed convex cone, but is not necessarily pointed.

We will now give Farkas lemma without proof and then discuss how it connects to our problem.

Lemma 8 (Farkas)**.** *Let $B \in \mathbb{R}^{n \times m}$, $C \in \mathbb{R}^{n \times p}$ be two matrices that define a cone*

$$K = \{Bx + Cy \mid y \in \mathbb{R}^p, x \in \mathbb{R}^m, x_i \geq 0, i = 1, \ldots, m\} \ .$$

For a vector $g \in \mathbb{R}^n$, we have either

1. *$g \in K$, or*
2. *there exists $p \in \mathbb{R}^n$ such, that*

$$g^T p < 0 \text{ and } B^T p \geq 0 \text{ and } C^T p = 0 \ ,$$

but not both.

This lemma is highly abstract, so we will look at one special case, based on our optimisation problem with linear equality constraints (35). For this, we rename x to λ in Farkas Lemma and set $B = 0$, $C = -A^T$ and $g = \nabla f(x)$. This gives rise to the cone $K = \{-A^T \lambda \mid \lambda \in \mathbb{R}^m\}$ and for this special case, Farkas Lemma reads that we have either

1. $\nabla f(x) \in K$, or

2. there exists $p \in \mathbb{R}^n$ such, that

$$\nabla f(x)^T p < 0 \text{ and } \quad -Ap = 0 \ ,$$

but not both.

The second condition means that at a point $x \in \mathcal{C}$ exists a direction p that is a descent direction ($\nabla f(x)^T p < 0$) which is also inside the tangent cone $T_{\mathcal{C}}(x)$ (as $Ap = 0$). Thus, by Theorem 8, x is not a critical point. Farkas Lemma now states that if this p does exist, we have $\nabla f(x) \notin K$. The reverse also holds and we can state that, if x is a critical point, then there exists a $\lambda \in \mathbb{R}^m$ such that $\nabla f(x) + A^T \lambda = 0$. This finally gives us a way to give concise optimality conditions for the equality constrained problem (34) as we summarise in the following Lemma:

Lemma 9. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable. Further, let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^n$. Consider the constrained optimisation problem*

$$\begin{aligned} &\min_x f(x) \\ &s.t.\ Ax + b = 0 \ . \end{aligned}$$

A point $x^* \in \mathbb{R}^n$ is a critical point according to Theorem 8 if and only if there exists $\lambda^* \in \mathbb{R}^m$ such, that

$$\begin{aligned} &\nabla f(x^*) + A^T \lambda^* = 0 \\ \wedge \quad &Ax^* + b = 0 \end{aligned} \tag{36}$$

The conditions are simple enough to use them as a stopping criterion for an algorithm. The norm $\|Ax + b\|$ measures the degree to which the equality constraints are violated by the point x. For an algorithm that only proposes feasible points, this will always be fulfilled up to numerical rounding errors, but we might find better convergence in algorithms that are allowed to query the function value $f(x)$ of points that are not feasible. The other condition can be turned into a norm $\|\nabla f(x) - A\lambda\|$, which measures how close the gradient is to the vector $A\lambda$ inside the cone. For a given x, if we set out to find the λ that minimises the norm, then this gives a measure for how close we are to the optimum. Alternatively, algorithms can keep track of both x and λ and find their joint optimum.

Example (Quadratic functions): We will look at the problem of optimising a strictly convex quadratic function under equality constraints. We pick $f(x) = \frac{1}{2} x^T Q x + g^T x$, Q symmetric positive definite and with this, problem (35) reads

$$\begin{aligned} \min_x \ & \frac{1}{2} x^T Q x + g^T x \\ \text{s.t. } & Ax + b = 0 \ . \end{aligned} \tag{37}$$

And since $\nabla f(x) = Qx + g$, we can rewrite (36) in matrix form as

$$\begin{pmatrix} Q & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} x^* \\ \lambda^* \end{pmatrix} = \begin{pmatrix} -g \\ -b \end{pmatrix} \ . \tag{38}$$

Thus, we can solve problem (37) simply by solving a system of equations. Unfortunately, the matrix is never positive definite, but invertible as long as $\text{rank}(A) = m$, i.e, all linear equalities are independent.

5.2 Algorithms for Linear equality constrained problems

In this section, we will adapt some of our algorithms to the inclusion of linear equality constraints, that is, the solution of problem (35). The idea is to adapt our algorithms such that we ensure that our step p_k proposed at position x_k is an element of the tangent cone, that is, $Ap_k = 0$. With this, we have for the new point $x_{k+1} = x_k + \alpha_k p_k$, when inserting into the constraint that

$$Ax_{k+1} + b = A(x_k + \alpha_k p_k) + b = Ax_k + b + \alpha_k A p_k = Ax_k + b \ .$$

Thus, when we choose x_0 as a feasible point, then all x_k are feasible. With this idea, it is possible to adapt Steepest Descent, Algorithm 3 and Newton's algorithm, Algorithm 4. Adapting BFGS, Algorithm 9 is more complicated, because with constraints, we can not ensure the Wolfe conditions as the optimum might never fulfil the curvature condition.

Data: x_0 fulfilling $Ax + b = 0$, $f : \mathbb{R}^n \to \mathbb{R}$, $A \in \mathbb{R}^{m \times n}$, $0 < c_1 < 1$, $0 < \rho < 1$
Result: Estimate of local minimum x_{k+1} of f with $x_{k+1} \in \mathcal{C}$
for $k = 0, \ldots,$ *until any stopping condition is fulfilled* **do**
  if $\nabla^2 f(x_k)$ *is positive definite* **then**
    $B = \nabla^2 f(x_k)$
  else
    Compute eigenvalues λ_i and eigenvector v_i of $\nabla^2 f(x_k)$;
    $B = \sum_{i=1}^{N} |\lambda_i| v_i v_i^T$;
  Solve $\begin{pmatrix} B & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} p_k \\ \lambda^* \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) \\ 0 \end{pmatrix}$;
  $\alpha_k = \text{backtrack}(x_k, p_k, 1.0, c_1, \rho)$;
  $x_{k+1} = x_k + \alpha_k p_k$;

Algorithm 10: Linearly equality constrained Newtons Algorithm

Newton's Algorithm. Adapting the Newton step is straight forward, as we have already done the most important work in our example for the linear equality constrained quadratic problem. All we have to do is to change the problem of minimising $m_k(p)$ to include that $p \in T_{\mathcal{C}}(x)$, that is it fulfils $Ap = 0$. The full problem description reads

$$\begin{aligned} \min_p \; & m_k(p) = \frac{1}{2} p^T \nabla^2 f(x_k) p + \nabla f(x_k)^T p \\ \text{s.t. } & Ap = 0 \ . \end{aligned}$$

This is an instance of problem (37) and we have already computed its solution. Aside from that, we perform the same steps as in Algorithm 4, including handling of indefinite matrices. The resulting algorithm is given in Algorithm 10.

Data: x_0 fulfiling $Ax + b = 0$, $f : \mathbb{R}^n \to \mathbb{R}$, $A \in \mathbb{R}^{m \times n}$, $0 < c_1 < 1$, $0 < \rho < 1$
Result: Estimate of local minimum x_{k+1} of f with $x_{k+1} \in \mathcal{C}$
$\beta_0 = 1$;
$M = I - A^T (AA^T)^{-1} A$;
for $k = 0, \ldots,$ *until any stopping condition is fulfiled* **do**
  $p_k = -M \nabla f(x_k)$;
  $\alpha_k = \text{backtrack}(x_k, p_k, \beta_k, c_1, \rho)$;
  $x_{k+1} = x_k + \alpha_k p_k$;
  $\beta_{k+1} = \frac{\alpha_k}{\rho}$;

Algorithm 11: Linearly equality constrained Steepest Descent

Steepest Descent. For Steepest Descent, the adaptation is a bit more elaborate. Without constraints, we would pick $p_k = -\nabla f(x_k)$. However, with constraints this will almost surely not be an element of the tangent cone. Thus, we have to pick a descent direction in the tangent cone, that is p_k must fulfil

$$p_k^T \nabla f(x_k) < 0 \text{ and } Ap_k = 0 \ .$$

However, there are an infinite number of vectors that fulfil these conditions and many of them will only lead to small progress. What we want is to pick the element in $T_{\mathcal{C}}(x_k)$ that is closest to $-\nabla f(x_k)$, that is it minimises the euclidean norm and fulfils

$$p_k = \arg \min_{p \in T_{\mathcal{C}}(x_k)} \frac{1}{2} \| p - (-\nabla f(x_k)) \|^2 \ .$$

This is another instance of our quadratic problem (37). The naive solution of this system is as expensive as the computation of the constrained Newton step. Using this naive solution in our implementation would remove one of the true upsides of Steepest descent, that it is much faster. Instead, we will use a direct analytic solution of the system, which leads to

$$p_k = -\left(I - A^T (AA^T)^{-1} A\right) \nabla f(x_k) \ .$$

This requires inversion of the matrix $AA^T \in \mathbb{R}^{m\times m}$. While this is potentially expensive, in many instances m is small. Further, as this matrix is constant, we can precompute $M = I - A^T(AA^T)^{-1}A$ at the start of the algorithm and then reuse it at each iteration. This leads to algorithm (11). The matrix M has quite an interesting interpetation: it is the solution of applying Gram-Schmidt Orthogonalisation to $-\nabla f(x_k)$ in order to make it orthogonal to all rows of the matrix A.

6 The Karush-Kuhn-Tucker Conditions

In the last section, we have started with a discussion of the first order necessary conditions of constrained optimisation problems. We derived the necessary first order conditions using the tangent cone $T_{\mathcal{C}}$, which uses a geometric description of the constraints. After this, we limited the scope to linear equality constraints and used Farkas' Lemma, Lemma 8, to derive *algebraic* optimality conditions. In this chapter, our goal is to generalise this result to the general constrained problem, that we restate here:

$$\begin{aligned} &\min_x f(x) \\ &\text{s.t. } c_i(x) = 0,\ i \in \mathcal{E} \\ &\wedge\ c_i(x) \leq 0,\ i \in \mathcal{I} \end{aligned} \tag{39}$$

Again, $c_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$ are constraint functions, which are partitioned in equality and inequality constraints. The equality constraints have $c_i(x) = 0$, and their indices are included in the set $\mathcal{E}$. The inequality constraints have $c_i(x) \leq 0$ and their indices are included in $\mathcal{I}$. We assume that all c_i are continuous differentiable functions.

We have already seen that inequality constraints do not constrain our directions of improvement as long as they are not fulfiled with equality due to continuity. Thus, when searching for an optimisation direction, if $c_i(x) < 0$, it can be safely discarded from our considerations. We will formalize this distinction with the introduction of the *active set* at x.

Definition 7 (Active Set)**.** *A constraint c_i is called active at x if $c_i(x) = 0$. The set of all active constraints*

$$\mathcal{A}(x) = \{i \in \{1, \ldots, m\} \mid c_i(x) = 0\} \ ,$$

is called the active set.

Equality constraints are always active, while inequality constraints only if $c_i(x) = 0$. We will now aim to find an algebraic description of the tangent cone. For this, we will define a new set, the feasible direction set

Definition 8 (Feasible Direction Set)**.** *The feasible direction set at x is the set of all vectors $p \in \mathbb{R}^n$ for which holds*

$$\nabla c_i(x)^T p = 0, i \in \mathcal{E}$$

and

$$\nabla c_i(x)^T p \leq 0, i \in \mathcal{I} \cap \mathcal{A}(x)$$

We call this set $\mathcal{F}(x)$

The feasible direction set is our proposal for the algebraic description of $T_{\mathcal{C}}(x)$. When we compare this to the linear equality constrained case of the

previous chapter, then we already saw that $T_{\mathcal{C}}(x) = \mathcal{F}(x)$, by Lemma 7. The feasible set extends this idea by also adding a condition for each active inequality constraint so that a p is only included if it is not an ascent direction on any active inequality constraint. The following result gives us hope that this might be correct

Lemma 10. *It holds*

$$T_{\mathcal{C}}(x) \subseteq \mathcal{F}(x)$$

Proof. To show this, let us pick any $p \in T_{\mathcal{C}}(x)$ at any $x \in \mathcal{C}$. Then, there exists a $u > 0$ and a $z : [0, u) \to \mathcal{C}$ such, that $z(0) = x$ and $z'(0) = p$. Now, pick any equality constraint, c_i, $i \in \mathcal{E}$. Since $z(\alpha) \in \mathcal{C}$ for $\alpha \in [0, u)$, we have $c_i(z(\alpha)) = 0$. We now take the Taylor expansion around 0 and take any $\alpha \in (0, u)$. We have

$$\begin{aligned}
& 0 = c_i(z(\alpha)) \\
\Leftrightarrow\ & 0 = c_i(z(0)) + \alpha \frac{\partial}{\partial \alpha} c_i(z(0)) + R(\alpha) \\
\Leftrightarrow\ & 0 = 0 + \alpha \nabla c_i(x)^T p + R(\alpha) \\
\Leftrightarrow\ & 0 = \nabla c_i(x)^T p + \frac{R(\alpha)}{\alpha}
\end{aligned}$$

Since $R(\alpha)/\alpha \to 0$ as $\alpha \to 0$ due to Taylor's theorem, we have $\nabla c_i(x)^T p = 0$ as $\alpha \to 0$. This shows the result for the equality constraints. For the inequality constraints, the same limit argument applies for any c_j, $j \in \mathcal{I} \cap \mathcal{A}(x)$ and it holds $\nabla c_j(x)^T p \leq 0$ □

This result formalises our intuition that we can only take directions that are orthogonal to all equality constraints, while for active inequality constraints we are additionally allowed to use directions that leave the constraint towards the side of the smaller constraint values. Unfortunately, the other direction, $\mathcal{F}(x) \subseteq T_{\mathcal{C}}(x)$ does not always hold as the next example shows

Example: Consider the equality constraint $c_1(x) = (a^T x + b)^2 = 0$. For a feasible point, we have $a^T x + b = 0$. But for the derivative of c_1 we have

$$\nabla c_1(x) = \underbrace{(a^T x + b)}_{0} a = 0 \ .$$

Thus, for all p we have $\nabla c_1(x)^T p = 0$, even though it might not be a feasible direction.

This example demonstrates the difference between the geometric and algebraic description of the tangent cone. The geometric description via $T_{\mathcal{C}}(x)$ only uses which points are feasible, and does not use the exact algebraic description of the constraints. In contrast, the algebraic description depends on that correct information is available through the gradients and function values of the

constraint functions. In the example above, the constraint was written in such a way that the gradient is meaningless.

This opens up the question how we can assess whether the algebraic description via the feasible direction set matches the tangent cone. This is a difficult task and there are several approaches. These approaches are called constraint qualifications, properties that the constraints must fulfil. One important constraint qualification is the linearly independent constraint qualification (LICQ)

Definition 9 (LICQ)**.** *Let $x \in \mathcal{C}$ and $\mathcal{A}(x)$ be the active set at x. We say that the linearly independent constraint qualification (LICQ) holds at x, if the vectors*

$$\nabla c_i(x),\ \forall i \in \mathcal{A}(x)$$

are independent.

In our previous example, the LICQ does not hold, because the zero-vector is linearly dependent. With it, it is possible to show the following result:

Lemma 11. *If the LICQ holds at x, then*

$$T_{\mathcal{C}}(x) \subseteq \mathcal{F}(x)\ .$$

This result requires the implicit function theorem to create a function z for any element of $\mathcal{F}(x)$. Due to the complexity and the little knowledge gain of this result, we skip the proof. Even though these results are encouraging, the LICQ is not the broadest or only relevant constraint qualification. Some very simple constraints do not fulfil it, even though one can show that a variant of Lemma 11 holds for them. Indeed we have already seen one example. In the last chapter, we have stated in Lemma 7 that any set of linear equality constraints, where the c_i are linear functions fulfils this property. This result did not require the linear equations to be independent, unlike LICQ. Still, the LICQ is important, because it describes a particularly simple case. For example, in the previous section, we still profited from linear independence of the constraint normals as otherwise we would not be able to invert some of the matrices. When formulating optimisation problems, one should strive for that the LICQ holds everywhere.

Finally, we have all the required ingredients to show the full algebraic first order sufficient conditions for problem (39). This is a result due to Karush, Kuhn and Tucker and is known as the KKT theorem

Theorem 9 (Karush-Kuhn-Tucker (KKT))**.** *Suppose that x^* is a local minimum of problem (39) and that the functions f and c_i, $i = 1, \dots, m$ are differentiable. Suppose further, that the LICQ holds at x^*. Then there exists a set of*

scalars $\lambda_i^* \in \mathbb{R}$, $i = 1, \ldots, m$ *and it holds*

$$\nabla f(x^*) + \sum_{k=1}^{m} \lambda_k^* \nabla c_k(x^*) = 0 \tag{40}$$

$$c_i(x) = 0, \ i \in \mathcal{E} \tag{41}$$

$$c_j(x) \leq 0, \ j \in \mathcal{I} \tag{42}$$

$$\lambda_j^* \geq 0, \ j \in \mathcal{I} \tag{43}$$

$$\lambda_j^* \cdot c_j(x) = 0, \ j \in \mathcal{I} \tag{44}$$

Before we proof the theorem, let us try to obtain a bit of intuition. We have already seen a variation of the first two lines last chapter in the special case for linear equality constraints, Lemma 9, and one can see that it is a special of the KKT. The last three lines are new and all of them are handling the inequality constraints. The third line is not controversial: x^* must fulfil the constraints to be an optimum. The last line is called the complementary condition. It says, that at least one of $c_j(x) = 0$ or $\lambda_j^* = 0$ must hold. This line is a result of the active set: we already noticed that inequality constraints only affect the tangent cone $T_{\mathcal{C}}(x^*)$ if the constraint is active, otherwise at x^* the problem would behave the same if we removed the constraint. Indeed, we can rephrase the condition easily as $\lambda_j^* = 0$ if $j \in \mathcal{I}$ and not $j \in \mathcal{A}(x^*)$. Thus, all that remains are the new λ_j^* terms for the inequality constraints in the first line, as well as line 4. In the proof, we obtain them through Farkas Lemma, but the fourth line can also be understood as active inequality constraints only ever need to prevent leaving the constraint line in one direction, unlike equality constraints where deviation to either $c_i(x^*) > 0$ or $c_i(x^*) < 0$ are not allowed.

Proof of Theorem 9. As x^* is a local minimum, the geometric first order conditions, Theorem 8, hold and we have

$$\nabla f(x^*)^T p \geq 0, \ \forall p \in T_{\mathcal{C}}(x^*) \ .$$

Since the LICQ holds at x^* by assumption, we have $T_{\mathcal{C}}(x^*) = \mathcal{F}(x^*)$ and thus

$$\nabla f(x^*)^T p \geq 0, \ \forall p \in \mathcal{F}(x^*) \ .$$

When we set $A_{\mathcal{E}}$ as the matrix with rows $\nabla c_i(x^*)$, $i \in \mathcal{E}$ and $A_{\mathcal{I}}$ as the matrix with rows $\nabla c_i(x^*)$, $i \in \mathcal{E} \cap \mathcal{A}$, the previous result is equivalent to

$$\nabla f(x^*)^T p \geq 0 \text{ and } -A_{\mathcal{E}} p = 0 \text{ and } -A_{\mathcal{I}} p \geq 0.$$

Next we will use Farkas Lemma. Farkas Lemma states that either there is a descent direction p (note the first inequality change):

$$\nabla f(x^*)^T p \leq 0 \text{ and } -A_{\mathcal{E}} p = 0 \text{ and } -A_{\mathcal{I}} p \geq 0.$$

or it holds

$$p \in K = \{-A_{\mathcal{I}} \lambda_{\mathcal{I}} - A_{\mathcal{E}} \lambda_{\mathcal{E}} \mid \lambda_{\mathcal{I}} \geq 0\} \ .$$

The vectors $\lambda_{\mathcal{I}}$ and $\lambda_{\mathcal{E}}$ take the roles of x and y in Farkas Lemma, respectively. Since x^* is a local minimum, we have $p \in K$ and there exist a pair of vectors $\lambda_{\mathcal{I}}$ and $\lambda_{\mathcal{E}}$ such, that $p = -A_{\mathcal{I}}\lambda_{\mathcal{I}} - A_{\mathcal{E}}\lambda_{\mathcal{E}}$.

We now define a vector $\lambda^* \in \mathbb{R}^m$. Its elements are set the following: if the ith constraint has row j in $A_{\mathcal{E}}$, then $\lambda_i^* = \lambda_{\mathcal{E},j}$. Similarly, if it has row j in $A_{\mathcal{E}}$, then $\lambda_i^* = \lambda_{\mathcal{I},j}$. Otherwise, if $i \notin \mathcal{A}(x^*)$, then $\lambda_i^* = 0$.

We now only need to show that this λ_i^* fulfils the KKT conditions above. The first condition (40) can be seen to be $p \in K$ rewritten with λ^*. The second and third conditions are fulfilled from x^* alone. The fourth condition is fulfilled as the respective elements are given by the respective elements of $\lambda_{\mathcal{I}}$ that are non-negative by Farkas Lemma. And the fifth condition holds as by construction of λ^*, all inequality constraints not in the active set have $\lambda_i^* = 0$. □

With this, we have now derived the full first order necessary conditions for constrained optimisation problems. As before for the unconstrained case, these conditions can not differentiate between optima and saddle-points and again, proper description of the *algebraic* second order conditions require advanced tools to derive them. We will omit these conditions as in practice the first order conditions are often sufficient to find a local optimum.

6.1 Example: Quadratic function with norm constraint

We are now going to present one example for the application of the KKT conditions. As an example we use the following quadratic function with non-linear equality constraint

$$\min_x \; f(x) = \frac{1}{2}x^T Q x + x^T g \tag{45}$$

$$\text{s.t. } \|x\|^2 \leq \Delta^2 \tag{46}$$

This means we search the minimum of a quadratic function for x that are not allowed to have a norm larger than $\Delta > 0$. It is important here that we put no condition of Q. Especially in can have negative eigenvalues, in which case the unbounded minimum of the function would not exist. First, we bring the constraint in standard form and multiply the constraint with the factor $1/2$ for notational convenience later on.

$$\min_x f(x) = \frac{1}{2}x^T Q x + x^T g \tag{47}$$

$$\text{s.t.} \qquad c(x) = \frac{1}{2}(\|x\|^2 - \Delta) \leq 0 \tag{48}$$

With this added factor of $1/2$, the gradient of the constraint has derivative

$$\nabla c(x) = x$$

And this means that at all x where the constraint is active (that means, $c(x) = 0$), the LICQ conditions are fulfilled as $\|\nabla c(x)\| = \Delta \neq 0$. Thus, we are allowed

to use the KKT conditions for optimality, which means that at a critical point x^*, there exists a $\lambda^* \in \mathbb{R}$ that fulfils

$$\begin{aligned} \nabla f(x^*) + \lambda^* x^* &= 0 \\ \|x^*\|^2 - \Delta &\leq 0 \\ \lambda^* &\geq 0 \\ \lambda^* \cdot (\Delta - \|x^*\|^2) &= 0 \end{aligned}$$

We can simplify the first line to yield:

$$\begin{aligned} &\nabla f(x^*) + \lambda^* x^* = 0 \\ \Leftrightarrow &Qx^* + g + \lambda^* x = 0 \\ \Leftrightarrow \ &(Q + \lambda^* I_n) x^* = -g \end{aligned}$$

The KKT conditions only provide information whether a pair of (x^*, λ^*) is a critical point, but not whether it is a local minimum. We will do this in the following lemma, which also shows how we can leverage the KKT conditions to derive meaningful results about local optima.

Lemma 12. *Let $x^* \in \mathbb{R}^n$ and $\lambda^* \in \mathbb{R}^n$ a pair of values that fulfil the KKT conditions of problem (45):*

$$\begin{aligned} (Q + \lambda^* I_n) x^* &= -g \\ \|x^*\|^2 - \Delta &\leq 0 \\ \lambda^* &\geq 0 \\ \lambda^* \cdot (\Delta^2 - \|x^*\|^2) &= 0 \end{aligned}$$

Then, x^* is a local optimum , if and only if

$$Q + \lambda^* I_n \text{ is positive semi-definite.}$$

Proof. The first KKT condition implies that x^* is a critical point of the function

$$h(x) = \frac{1}{2} x^T (Q + \lambda^* I_n) x + x^T g = f(x) + \frac{\lambda^*}{2} x^T x \ ,$$

which can be shown by differentiating and setting the gradient to zero. For the "if" part, assume $Q + \lambda^* I_n$ is positive semi-definite. Then, due

to the first KKT condition, we can write $x = x^* + p$

$$\begin{aligned} h(x^* + p) &= \frac{1}{2}(x^* + p)^T(Q + \lambda^* I_n)(x^* + p) + (x^* + p)^T g \\ &= \frac{1}{2}x^{*T}(Q + \lambda^* I_n)x^* + x^{*T} g + \frac{1}{2}p^T(Q + \lambda^* I_n)p \\ &+ p^T \underbrace{((Q + \lambda^* I_n)x^* + g)}_{0} \\ &= h(x^*) + \frac{1}{2}p^T(Q + \lambda^* I_n)p \end{aligned}$$

As $Q + \lambda^* I_n$ is positive semi-definite, we have $p^T(Q + \lambda^* I_n)p \geq 0$, which shows that x^* the global minimiser of h.
With this and the definition of h, we have for any feasible x

$$f(x) \geq f(x^*) + \frac{\lambda^*}{2}(x^{*T}x^* - x^T x).$$

We now use the complementary condition of the KKT theorem, $\lambda^* \cdot (\Delta - \|x^*\|^2) = 0$ to obtain

$$\begin{aligned} f(x) &\geq f(x^*) + \frac{\lambda^*}{2}(x^{*T}x^* - x^T x) \\ &= f(x^*) + \frac{\lambda^*}{2}(x^{*T}x^* - \Delta^2) + \frac{\lambda^*}{2}(\Delta^2 - x^T x) \\ &= f(x^*) + \frac{\lambda^*}{2}(\Delta^2 - x^T x) \end{aligned}$$

Since $x^T x \leq \Delta^2$, we have that x^* is global minimum of problem (45).

For the "only if" part, we need to show that

$$p^T(Q + \lambda^* I_n)p \geq 0, \forall p \in \mathbb{R}^n.$$

Assume that x^* is the global optimum of problem (45) and take any point x with $x^T x = x^{*T}x^* \leq \Delta^2$. Then, it holds again

$$f(x) \geq f(x^*) + \frac{\lambda^*}{2}\underbrace{(x^{*T}x^* - x^T x)}_{0} \ .$$

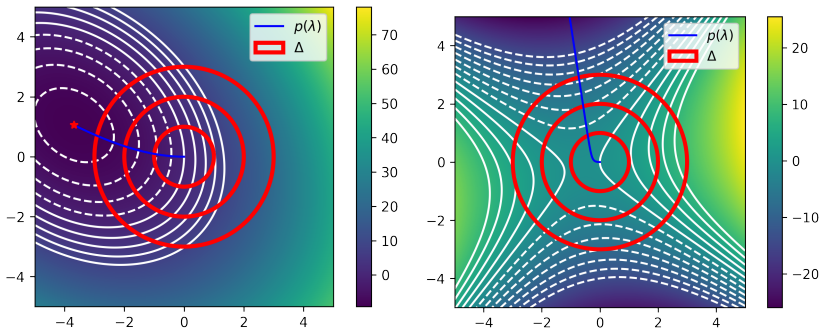
When we simplify this equation and use the first KKT condition to derive and insert $g = -(q + \lambda^* I_n)x^*$, we obtain

$$(x - x^*)^T(Q + \lambda^* I_n)(x - x^*) \geq 0.$$

If the previous result holds, then it follows that for any $p = \alpha \cdot (x - x^*)$ holds

$$p^T(Q + \lambda^* I_n)p \geq 0.$$

Since x is chosen arbitrarily on the sphere, p covers the full $\mathbb{R}^n$ and thus $Q + \lambda^* I_n$ is positive semi-definite. □



(a) Q positive definite (b) Q indefinite

Figure 7: Visualisation of the path $p(\lambda)$ for two different problems in 2D. Left: Q is positive definite and an unconstrained global minimum c^* exists (red star). Depending on Δ (red circles) a different solution along the path (blue curve) is taken. for Δ larger than the distance to the optimum, the optimal solution is c^*. Right: If Q is indefinite, there is no global optimum. While there are several plausible paths (up or down), one of them leads to lower function values. Note that for very small Δ, the direction of the path changes a lot as for small Δ the gradient dominates the function value.

This result shows conditions for a given pair of (x^*, λ^*) to be global optima, but it does not provide a way to find a solution. Still, we can use it to guide us on our search. We can analyse different cases:

Case 1: When Q is positive definite, we can try $\lambda = 0$. Then, we have from the first KKT condition that

$$x = -Q^{-1}g$$

which is the unconstrained minimum. If for this holds $\|x\| \leq \Delta$, then the unconstrained minimum is feasible and the complementary conditions holds, thus x^* is the global optimum.

Case 2: Either Q is not positive definite, or the unconstrained optimum of Case 1 is not feasible. Try $\lambda > \max\{0, -\lambda_1(Q)\}$, where $\lambda_1(Q)$ is the smallest eigenvalue of Q. This condition sets $\lambda > -\lambda_1(Q)$ if Q has a negative eigenvalue and $\lambda > 0$ otherwise. In this case, $Q + \lambda I_n$ is positive definite. Moreover, by complementary condition, as $\lambda > 0$ we now require that $\|x\| = \Delta$, thus we are interested in finding a solution on the boundary.

Due to these conditions, $Q + \lambda I_n$ is invertible and we can use the first KKT condition and find

$$x = -(Q + \lambda I_n)^{-1}g$$

Depending on λ, the obtained x follows a path

$$p(\lambda) = -(Q + \lambda I_n)^{-1}g$$

We visualise $p(\lambda)$ in a simple 2D problem for two cases where Q is either positive definite, or indefinite in Figure 7 The idea is now that we can pick a λ such, that $\|p(\lambda)\| = \Delta$. But to derive a numerical approach for finding λ, we need to know how the path behaves with it.

We now want to analyse the length of this path as we change λ. We will derive that

$$\lim_{\lambda \to \infty} \|p(\lambda)\| = 0$$

and, providing that g is not orthogonal to the eigenvector belonging to the smallest eigenvalue of Q, we have

$$\lim_{\lambda \to -\lambda_1(Q)} \|p(\lambda)\| \to \infty$$

Otherwise, the path length is bounded.

Derivation: Let $Q = UDU^T$ be the eigenvalue decomposition of Q, with $\lambda_i(Q)$ being the eigenvalues on the diagonal of the diagonal matrix D. It holds then that

$$(Q + \lambda I_n)^{-1} = (UDU^T + \lambda UU^T)^{-1} = U(D + \lambda I_n)^{-1}U^T \ .$$

For the squared norm, we can then derive

$$\begin{aligned}
\|p(\lambda)\|^2 &= p(\lambda)^T p(\lambda) \\
&= \underbrace{g^T U}_{y}(D + \lambda I_n)^{-1} \underbrace{U^T U}_{I_n}(D + \lambda I_n)^{-1}U^T g \\
&= y^T((D + \lambda I_n)^{-1})^2 y \\
&= \sum_{i=1}^{n} \frac{y_i^2}{(\lambda_i(Q) + \lambda)^2}
\end{aligned}$$

For the results: as $\lambda \to \infty$, the denominator in each term will grow towards ∞ and thus the norm will shrink to 0. Otherwise, as $\lambda \to -\lambda_1(Q)$, the term $(\lambda_1(Q) + \lambda)^2 \to 0$. if the corresponding $y_i^2 \neq 0$, this means that the path length goes to infinity. Otherwise, it remains bounded.

This result is encouraging. If the unconstrained optimum of Case 1 was not feasible, this means that when we choose λ large enough, then we will eventually find a point with $\|p(\lambda)\| = \Delta$. Otherwise, if Q is infeasible, then we need to shorten the path length. In this case, provided that g is not orthogonal to the eigenvector belonging to the smallest eigenvalue, then as $\lambda \to -\lambda_1(Q)$, we will eventually find a solution that is short enough.

So, what about the case that g is orthogonal to the eigenvector belonging to the smallest eigenvalue? In this case, either the bounded path length is longer than Δ, or there does not exist a solution with $\lambda > \max\{0, -\lambda_1(Q)\}$.

Case 3, the hard case: If the first two cases do not find a solution, then we have $\lambda = \max\{0, -\lambda_1(Q)\}$ and $Q + \lambda I_n$ has smallest eigenvalue zero. Moreover, g is orthogonal to the eigenvector belonging to this eigenvalue. In this case, the path does not exist and there might be multiple solutions.

6.2 An algorithm for solving the Quadratic function with norm constraint

Data: $Q \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, radius Δ, precision $\epsilon > 0$
Result: Optimal feasible x^*
Let $p(\lambda) = -(Q + \lambda I_n)^{-1} g$;
if $\lambda_1(Q) > 0$ *and* $\|p(0)\| \leq \Delta$ **then**
  $x^* = p(0)$ and stop;
$l \leftarrow \max\{0, -\lambda_1(Q)\}$;
$u \leftarrow \max\{0, -\lambda_1(Q)\} + 1$;
while $\|p(u)\| > \Delta$ **do**
  $l \leftarrow u$;
  $u \leftarrow 2 \cdot u$;
while *True* **do**
  $\lambda' = \frac{1}{2}(l + u)$;
  $p' = p(\lambda')$;
  if $\|p'\| < \Delta$ *and* $|\|p'\| - \Delta| < \epsilon$ **then**
    $x^* = p'$ and stop;
  if $\|p'\| > \Delta$ **then**
    $l \leftarrow \lambda'$;
  else
    $u \leftarrow \lambda'$;

Algorithm 12: Solver for the quadratic function with norm constraint

We will now discuss an algorithm for solving the problem. For this, we will assume that we are not in the hard case, thus, either Case 1 or Case 2 will find a solution. Numerically, the hard case is very unlikely to occur, unless this case is constructed explicitly. The algorithm we propose is given in Algorithm 12. Unlike last week, in this week the algorithm will propose x which are not feasible during evaluation, but converge to a feasible point. Moreover, the solution will not provide a solution where the constraint is active, when the unconstrained solution is not feasible, but just a point that is very close to the boundary.

The algorithm works by using a bisection search for λ. At the beginning, Case 1 is tested, that means whether the unconstrained optimum is feasible. If this is the case, the algorithm terminates immediately with the solution. Otherwise, we know that the solution must lie on the constraint boundary.

To search the correct λ, the algorithm then creates an initial bracket (l, u) in which a suitable λ can be found. For this, the algorithm tries increasingly larger values of u, until a point is found for which $\|p(u)\| < \Delta$, otherwise u is

increased by a factor of two. In each iteration l is updated to the old value of u to save computation time. Due to this, and the initialisation of l, it holds afterwards that $\|p(l)\| > \Delta$ and $\|p(u)\| < \Delta$.

Finally, the algorithm enters the main loop, where in each iteration the midpoint of the interval is queried. If the point is feasible and its length within ϵ of the true length, the algorithm terminates. Otherwise, if the point is too large, we choose it as the new lower bound, and otherwise as the new upper bound. This way the interval shrinks by a factor of 2 in each iteration, and thus, due to continuity of the $\|p(\lambda)\|$, the midpoint will eventually fulfil the stopping criterion.

It is interesting that this algorithm is so simple, even though in the case of indefinite Q in high dimensions, there are many local minima along the boundary. Even already for $n = 2$, as in Figure 7, we had two local optima (at the top and bottom of the circle) and using a gradient descent solution starting from a point on the boundary in the lower half will not end up at the global optimum at the upper path. These local optima fulfil the KKT conditions, but for the point in the lower half, $Q + \lambda I_n$ is not positive definite. If we change the norm to something else, like the L1-norm, $\sum_{i=1}^{n} |x_i| \leq \Delta$, we do not have this nice sufficient condition for the global optimum anymore and the best one can do is use exhaustive search to find the global optimum.

7 Trust Region Optimisation

Since the beginning of this course, we have been discussing line-search methods. In these methods, we first pick a step direction p_k using a local model $m_k(p) \approx f(x_k + p)$ and afterwards perform a line-search along the ray $x_k + \alpha p_k$ to obtain the length of the step. While the methods we used differed in the details around this basic methodology, we never investigated alternative approaches.

> **Note:** This section brings together knowledge from all prior sections in this course. If necessary, go back to the referenced sections and refresh your knowledge about the core ideas when they are mentioned.

Trust-region methods turn the basic approach of line-search methods around. In trust-region methods, we first pick a maximum steplength we are willing to take and then use our local model to find the best step that is not longer than this steplength. The intuition of this approach is that of a region of trust: we know that m_k approximates f well only in a small area around x_k. Outside this area the information we gain from m_k about f can be misleading. For example, if the local model predicts a global minimum far away from the current point, then we know that this prediction is very likely wrong - and why should we care about locating a minimum that we know is wrong? Instead, we can impose a restriction on the maximum steplength and then compute which step would be the best under this limitation. If our model performs well and obtains an improvement in function value, then we gain more trust in it and allow longer steps. On the other hand, if the model produced misleading information, we reduce the maximum steplength further.

Trust-region methods compute the optimal step by solving problems of the form

$$p_k = \arg\min_p m_k(p), \text{ s.t. } \|p\| \leq \Delta_k \ . \tag{49}$$

The set of steps p that fulfil the constraint $\|p\| \leq \Delta_k$ is called the trust-region. We have already seen this problem in Section 6.1, where we added this constraint to a quadratic function and developed an algorithm for finding the optimal solution. In Figure 7 we visualised the path of optimal solutions depending on the maximally allowed steplength Δ[2]. The important observation is that this path is not a straight line. If Δ is small, the direction chosen by the path will be similar to the gradient direction $\nabla m_k(0)$: for very small steps, the steepest descent direction leads to the largest improvement. As steps are allowed to become longer, the path curves more and more towards the critical point or the direction of the eigenvector belonging to the largest negative eigenvalue, if that exists. Indeed, these two direction can become almost orthogonal, as we have seen in Figure 7b. Therefore, choosing the steplength before the step direction can have an advantage, as when we expect successful steps to be much shorter

[2] We wrote the path depending on λ. However, for each λ there exists Δ for which it is the optimal solution.

than the Newton step, the improvement by taking the Newton step direction can be smaller than the steepest descent direction.

Moreover, using a trust region solves the biggest problem of Newtons method: when the Hessian $\nabla^2 f(x_k)$ is not positive definite, the Newton step does not exist. However, as the trust region problem (49) confines the solution within a closed bounded set, the model always obtains a minimum within the trust-region.

Data: $x_0 \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $\Delta_0 > 0$, $\eta \in (0, \frac{1}{4})$
Result: Estimate of local minimum x_{k+1} of f
for $k = 0, \ldots,$ *until any stopping condition is fulfilled* **do**
  Solve $p_k = \arg\min_p m_k(p)$, s.t. $\|p\| \leq \Delta_k$;
  $\rho_k = \dfrac{f(x_k + p_k) - f(x_k)}{m_k(p_k) - m_k(0)}$;
  if $\rho_k > \eta$ **then**
    $x_{k+1} = x_k + p_k$;
  else
    $x_{k+1} = x_k$;
  if $\rho_k < \frac{1}{4}$ **then**
    $\Delta_{k+1} = \frac{1}{4}\|p_k\|$;
  else
    if $\rho_k > \frac{3}{4}$ *and* $\|p_k\| = \Delta_k$ **then**
      $\Delta_{k+1} = 2\Delta_k$;
    else
      $\Delta_{k+1} = \Delta_k$;

Algorithm 13: Basic trust-region algorithm

We will now introduce the basic template for a trust-region algorithm, see Algorithm 13. This algorithm looks a lot more complicated, then the basic line-search algorithm, Algorithm 1. This is because this algorithm is much more detailed, for example it already includes a quality measure of the model, and an adaptation of the trust-region bound. We will now discuss the parts of the algorithm step by step and then introduce a detailed comparison to the line-search procedure.

In each iteration of the while loop, the algorithm first computes the optimal step of our chosen model within the trust-region, using equation (49). Afterwards, we compute a quality measure ρ_k,

$$\rho_k = \frac{f(x_k + p_k) - f(x_k)}{m_k(p_k) - m_k(0)} \quad . \tag{50}$$

The quality measure evaluates the new point on the real function and computes the improvement in function value. This value is divided by the predicted improvement by the model. As we minimise, this improvement in the model will always be negative, and thus $\rho_k > 0$ indicates that the point lead to an improvement on the real function, as predicted by the model. If $\rho_k \geq 1$, then

we have that improvement on the true function was even more than we predicted in the model, which is a very good outcome.

The performance measure ρ_k is first used to check whether the algorithm performs a step. Here, a step is only accepted if $\rho_k > \eta$, otherwise the algorithm uses in the next iteration the same point as in the current iteration. We will see that this condition is akin to the sufficient-decrease condition (4), the first part of the Wolfe conditions.

After the step has been performed, the algorithm updates Δ_k. The intuition of these steps is that if the model predicted the improvement well, that is ρ_k is large, we increase the trust-region radius – but only if p_k was on the boundary of the trust-region. This is because if the model predicts the function very well and p_k is on the boundary, then the trust region might limit our progress and increasing Δ might lead to faster improvement. Otherwise, if p_k is not on the boundary, this means that the Newton step was accepted and thus the trust region was not even part of the algorithm. In this case, we have no information about the trust-region boundary and changing it has no effect.

If the model predicted the function value poorly, that is ρ_k is small or even negative, then this indicates that the model is misleading and we should decrease its trust-region. We set the new value as a fraction of $\|p_k\|$. This takes into account that p_k might be inside the trust region - and thus if the Newton step is much smaller than Δ_k, decreasing Δ_k by a factor might not have an effect on the computed step in the next iteration. The exact values by how much Δ_k is increased or decreased are arbitrary and could be chosen differently, but it makes sense to decrease more aggressively than increasing, in order to get as many steps accepted as possible.

To get a better intuition about the trust-region approach, we will analyse the progress measure ρ_k defined in equation (50) and relate it to the sufficient decrease condition (4). We will obtain the following result:

Lemma 13. *Let $B_k \in \mathbb{R}^{n\times n}$. Let p_k be a step obtained from Algorithm 13 using model*

$$m_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T B_k p \ .$$

Further assume that $\rho_k > \eta$. Then the step $x_{k+1} = x_k + p_k$ fulfils the sufficient decrease condition

$$f(x_{k+1}) \leq f(x_k) + c_1 p^T \nabla f(x_k)$$

with $c_1 = \frac{\eta}{2}$.

Before we proof this lemma, let us see what it tells us about Algorithm 13. The result tells us that if a step is accepted, that is $x_{k+1} = x_k + p_k$, then the step fulfills the sufficient decrease condition. This was the same condition that we used for backtracking linesearch based algorithms, Algorithm 3&4, to accept a step. If a point did not fulfil the sufficient decrease condition, then we would discard it and half the steplength. Taking a look at Algorithm 13, this is also exactly what happens here. Since $\eta < 1/4$, a step that has $\rho_k < \eta$ will always use the branch that decreases the radius of the trust-region, and thus this part

of the algorithm mimics backtracking line-search, except that the points do no longer lie on a line but on a curved path.

Proof of Lemma 13. From the definition of ρ_k we have

$$\begin{aligned} \rho_k &= \frac{f(x_k + p_k) - f(x_k)}{m_k(p_k) - m_k(0)} \\ \Leftrightarrow f(x_k + p_k) - f(x_k) &= \rho_k(m_k(p_k) - m_k(0)) \\ \Leftrightarrow \qquad f(x_k + p_k) &= f(x_k) + \rho_k(m_k(p_k) - m_k(0)) \ . \end{aligned} \tag{51}$$

Next, we find a lower bound for $m_k(p_k) - m_k(0)$. The trust region problem (49) has the form required by Lemma 12 and therefore there exists a λ that fulfills the KKT conditions:

$$\begin{aligned} (B_k + \lambda I_n)p_k + \nabla f(x_k) &= 0 \\ \|p_k\|^2 - \Delta &\leq 0 \\ \lambda &\geq 0 \\ \lambda \cdot (\Delta^2 - \|p_k\|^2) &= 0 \end{aligned}$$

We can use the KKT conditions to rewrite the model improvement:

$$\begin{aligned} m_k(p_k) - m_k(0) &= \nabla f(x_k)^T p_k + \frac{1}{2} p_k^T B_k p_k \qquad (52) \\ &= \nabla f(x_k)^T p_k + \frac{1}{2} p_k^T B_k p_k - \underbrace{\frac{\lambda}{2} \cdot (\Delta^2 - \|p_k\|^2)}_{0} \\ &= p_k^T \nabla f(x_k) + \frac{1}{2} p_k^T (B_k + \lambda I_n) p_k - \frac{\lambda}{2}\Delta^2 \\ &= \frac{1}{2} p_k^T \left(\nabla f(x_k) + \underbrace{(B_k + \lambda I_n)p_k + \nabla f(x_k)}_{0} \right) - \frac{\lambda}{2}\Delta^2 \\ &= \frac{1}{2} p_k^T \nabla f(x_k) - \frac{\lambda}{2}\Delta^2 \leq \frac{1}{2} p_k^T \nabla f(x_k) \ . \qquad (53) \end{aligned}$$

During the derivation, we used the complementary condition to introduce λ and then reordered the terms such, that the first KKT condition emerged. In the last step we used tat the last term is negative and can be discarded to obtain a lower bound. We can now insert our result in (51) and since $\rho_k > \eta$ by assumption, we arrive at:

$$\begin{aligned} f(x_k + p_k) &= f(x_k) + \rho_k \underbrace{(m_k(p_k) - m_k(0))}_{\leq 0} \\ &\leq f(x_k) + \eta(m_k(p_k) - m_k(0)) \\ &\leq f(x_k) + \frac{\eta}{2} p_k^T \nabla f(x_k) \end{aligned}$$

□

General convergence proofs for trust-region methods are difficult to obtain. This is because the steps only fulfil the sufficient increase condition and not the other part of the Wolfe conditions, the curvature condition. This is also not something that one can easily add. Even though Algorithm 13 includes a mechanic to increase the trust region radius Δ arbitrarily, there is an upper limit to the steps: if the model uses positive definite B_k, then the step can not become longer than the Newton step. Thus, if the Newton step direction is too short, trust-region methods have no way to increase it further. Convergence proofs for trust-region methods therefore require stronger assumptions on the function for convergence.

Another problem for the proofs is that the solution to problem (49) is not easy to use. It is defined via the KKT conditions, which involve the unknown quantity λ, and thus measuring the progress on $m_k(p_k) - m(0)$ required for ρ_k is difficult. We will therefore introduce a lower bound that does not depend on λ. Even though we will not conduct full convergence proofs about trust-region methods, this result is necessary for them to show convergence, and any trust-region variant that chooses a step that achieves a lower function value on the model than this will converge on suitably chosen functions.

Lemma 14. *Let p_k be a solution to problem (49) using model*

$$m_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T B_k p \ .$$

Further, let β be the largest absolute eigenvalue of B_k. Then it holds

$$m_k(p_k) - m(0) \leq -\frac{1}{2} \|\nabla f(x_k)\| \min \left\{ \Delta, \frac{\|\nabla f(x_k)\|}{\beta} \right\} \ .$$

Proof. Let us find the minimum of m_k within the trust-region in the direction of the steepest descent direction, that is the minimum along the line $-\alpha \nabla f(x_k)$. The minimum is achieved for

$$\alpha^* = \arg\min_{\alpha} m_k(\alpha \nabla f(x_k)), \text{ s.t. } \|\alpha \nabla f(x_k)\| \leq \Delta_k \ .$$

It is easy to see that it holds

$$m_k(p_k) - m(0) \leq m_k(-\alpha^* \nabla f(x_k)) - m(0).$$

Thus, we only need to find α^* and bound that progress.
Inserting our definition of m_k and simplifying, the problem transform to

$$\begin{aligned} \min_{\alpha} \ & -\alpha \|\nabla f(x_k)\|^2 + \alpha^2 \frac{1}{2} \nabla f(x_k)^T B_k \nabla f(x_k) \\ \text{s.t. } & \|\alpha\| \leq \frac{\Delta_k}{\|\nabla f(x_k)\|} \end{aligned}$$

We first compute the unconstrained critical point of the problem. This

gives

$$\alpha' = \frac{\|\nabla f(x_k)\|^2}{\nabla f(x_k)^T B_k \nabla f(x_k)}$$

By second order sufficient condition, this is only a minimum when $\nabla f(x_k)^T B_k \nabla f(x_k) > 0$. Thus, we have two cases:

Case 1: $\alpha' \leq \frac{\Delta_k}{\|\nabla f(x_k)\|}$ and $\nabla f(x_k)^T B_k \nabla f(x_k) > 0$. In this case, we have $\alpha^* = \alpha'$, which leads to the progress

$$\begin{aligned} m(-\alpha^* \nabla f(x_k)) - m(0) &= -\frac{1}{2} \frac{\|\nabla f(x_k)\|^4}{\nabla f(x_k)^T B_k \nabla f(x_k)} \\ &\leq -\frac{1}{2} \frac{\|\nabla f(x_k)\|^2}{\beta} \\ &= -\frac{1}{2} \|\nabla f(x_k)\| \min\left\{\Delta, \frac{\|\nabla f(x_k)\|}{\beta}\right\} \end{aligned}$$

And the inequality was achieved by bounding $p^T B_k p \leq \beta \|p\|^2$. The last step is true because of the trust-region bound.

Case 2: If the conditions of case 1 do not hold, the solution must on the boundary and we have

$$\alpha^* = \frac{\Delta_k}{\|\nabla f(x_k)\|} \ .$$

If we have that $\nabla f(x_k)^T B_k \nabla f(x_k) \geq 0$, the progress can be bounded easily:

$$\begin{aligned} m(-\alpha^* \nabla f(x_k)) - m(0) &= -\Delta \|\nabla f(x_k)\| + \underbrace{\frac{1}{2} \frac{\Delta^2 \nabla f(x_k)^T B_k \nabla f(x_k)}{\|\nabla f(x_k)\|^2}}_{\leq 0} \\ &\leq -\Delta \|\nabla f(x_k)\| \end{aligned}$$

Which is smaller than the quantity we aim to proof.
Otherwise, we have that the unconstrained optimum is a minimum but not feasible and thus

$$\alpha' = \frac{\|\nabla f(x_k)\|^2}{\nabla f(x_k)^T B_k \nabla f(x_k)} > \frac{\Delta}{\|\nabla f(x_k)\|} \ .$$

Inverison of both sides gives

$$\frac{\nabla f(x_k)^T B_k \nabla f(x_k)}{\|\nabla f(x_k)\|^2} < \frac{\|\nabla f(x_k)\|}{\Delta} \ .$$

This can again be inserted into the progress, leading to:

$$m(-\alpha^* \nabla f(x_k)) - m(0) \leq -\frac{1}{2}\Delta \|\nabla f(x_k)\|$$

□

which also fulfills the bound.

With this, we could show local and global convergence properties. However, due to the missing curvature conditions, these properties always require further strong assumptions on the functions and then depending on the assumptions the details of the proof vary. Instead of showing further convergence results, we will just sketch the basic proof idea.

The key idea of all proofs is to introduce additional conditions on the function that allows our algorithm to make enough progress. The biggest hurdle for this proof is that we have to show that we do not run into a case, where Δ_k becomes small so quickly that the algorithm stops before reaching the optimum. To be more exact, we need to show the existence of a bound $\bar{\Delta}\|\nabla f(x_k)\|$, such that for all $\Delta_k \leq \bar{\Delta}\|\nabla f(x_k)\|$ the proposed step has $\rho_k > \frac{1}{4} > \eta$. In this case, the step is accepted and Δ_{k+1} is increased. As in these cases, the proposed step is on the boundary of the trust-region, this establishes that steps cannot stay arbitrarily small, as eventually Δ_k grows large enough such that $\|p_k\| \approx \bar{\Delta}\|\nabla f(x_k)\|$.

With this, we can now define a small local neighbourhood around some x_k that does not contain a critical point. We define those neighbourhoods such, that the norm of the gradient is bounded from below, $\|\nabla f(x_k)\| \geq u$ and therefore there exist an absolute lower bound for step lengths to be accepted. From this one can establish that the algorithm will eventually leave the neighbourhood towards points with smaller function value. As this holds for all neighbourhoods that do not include a critical point, the algorithm will eventually converge to a minimum.

Finally, it is also possible to show local convergence results similar to the ones we obtained for Newton's method. Indeed, if we set $B_k = \nabla^2 f(x_k)$, then it is possible to show that when the algorithm converges to a minimum, then eventually it will converge Q-quadratically and all steps will stay within the trust-region. This result is an exact analogue of local Q-quadratic convergence of Newtons method, where we showed that close to the optimum, steplength $\alpha_k = 1$ suffices.

7.1 Quasi-Newton for Inexact Trust-Region-Methods

We now set out to devise an efficient real-world quasi-Newton implementation of the basic trust-region algorithm. We discussed the benefits of quasi-Newton methods in Section 4: they provide partial Hessian information, without requiring to compute the Hessian, and they provide an efficient way for computing the approximate Newton step. This especially holds for the BFGS algorithm

which produced *positive definite* Hessian approximations, and thus due to the analytic inverse update rule of this approximation, computing a step was not only faster, but also easier to implement than even Newton's method. Experimentally, the costs of this in terms of convergence was often negligible. This allows quasi-Newton methods to scale up to very large problem sizes.

Data: $x_0 \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $\Delta_0 > 0$, $\eta \in (0, \frac{1}{4})$, $\epsilon_B > 0$
Result: Estimate of local minimum x_{k+1} of f
$B_0 = I_d$;
for $k = 0, \ldots,$ *until any stopping condition is fulfilled* **do**
  Set $m_k(p) = \frac{1}{2} p^T B_k p + \nabla f(x_k)^T p$;
  Solve $p_k = \arg\min_p m_k(p)$, s.t. $\|p\| \leq \Delta_k$;
  $\rho_k = \dfrac{f(x_k + p_k) - f(x_k)}{m_k(p_k) - m_k(0)}$;
  if $\rho_k > \eta$ **then**
    $x_{k+1} = x_k + p_k$;
  else
    $x_{k+1} = x_k$;
  $y_k = \nabla f(x_k + p_k) - \nabla f(x_k)$;
  $q_k = y_k - B_k p_k$;
  if $|p_k^T q_k| > \epsilon_B \|p_k\| \|q_k\|$ **then**
    $B_{k+1} = B_k + \frac{q_k q_k^T}{q_k^T p_k}$;
  if $\rho_k < \frac{1}{4}$ **then**
    $\Delta_{k+1} = \frac{1}{4}\|p_k\|$;
  else
    if $\rho_k > \frac{3}{4}$ *and* $\|p_k\| = \Delta_k$ **then**
      $\Delta_{k+1} = 2\Delta_k$;
    else
      $\Delta_{k+1} = \Delta_k$;

Algorithm 14: Exact SR1 Trust-Region algorithm

Looking at the benefits that can be provided by quasi-Newton approaches, it is questionable whether a trust-region approach can ever be competitive without also making use of at least some of these properties. A naive approach would be to just replace the model B_k by the BFGS approximation, using Lemma 6. However, this approximation requires that the curvature condition holds at the chosen step, as we have shown in Section 4.3, which we cannot enforce in our algorithm. Thus, we cannot use the BFGS approximation.

During derivation of BFGS, we have investigated a simpler update, the symmetric rank-1 (SR1) update in Lemma 5, which has the form

$$B_{k+1} = B_k + \frac{(y_k - B_k p_k)(y_k - B_k p_k)^T}{(y_k - B_k p_k)^T p_k} \quad ,$$

where p_k is the last step and $y_k = \nabla f(x_k + p_k) - \nabla f(x_k)$. This update did not

enforce positive definiteness of B_{k+1} and in a trust-region method, this is also no longer necessary.

The SR1 update has to be used with some care as the denominator $(y_k - B_k p_k)^T p_k$ might get very close to zero and thus we risk dividing by zero and other numerical instabilities. These problems always appear when $y_k - B_k p_k$ is almost orthogonal to our current step. To prevent these numerical instabilities, we will skip steps when

$$|p_k^T (y_k - B_k p_k)| \leq \epsilon_B \|p_k\| \|y_k - B_k p_k\| \ ,$$

Here, $\epsilon_B > 0$ is a small constant.

Direct application of these ideas to the basic trust-region method leads to Algorithm 14. We define m_k as the second order model using the SR1 approximated Hessian B_k and use this to compute p_k. Afterwards, we update the matrix B_k using the newly obtained gradient information. Otherwise, the algorithm stays the same. Note that we always update the hessian, even if the proposed step just got discarded: If our local approximation of the Hessian does not approximate the true function well, then very likely our estimate of the gradient at the proposed point was wrong. In that case, adding the gradient information will prevent us from suggesting this point again.

This algorithm fulfils a part of the requirements that we had for quasi-newton methods. Computing the Hessian is no longer necessary, and instead, it is replaced by an approximation scheme. However, we are still lacking a fast algorithm for computing p_k using this approximation. Unlike in BFGS, the inverse of B_k alone is not sufficient, as solving the trust-region subproblem using Algorithm 12 requires the inverse of $B_k + \lambda I_n$ for various choices of λ. Moreover, as B_k has a negative eigenvalue, we need to ensure that λ is large enough, so that $B_k + \lambda I_n$ is positive definite.

This makes it difficult to speed up our existing solver for the trust-region sub-problem, However, we can use that trust-region methods do not need to find the optimal solution of the trust-region problem. It is sufficient when the proposed p_k at least improves the model by the amount in Lemma 14. This progress is achieved by finding the optimal steepest descent steplength on m_k, taking the trust-region constraint into account. Any improvement over this will help the algorithm, however the more effort we invest into finding better p_k the more we gain. A very simple approach could be to compute the optimal steepest descent step together with the approximate Newton step $-B_k^{-1} \nabla f(x_k)$. If B_k is not positive definite, then the computed point will be a saddle-point. However, if it is feasible, it might still provide a better function value than the steepest descent step. And if B_k is positive definite and the point is feasible, then we obtain the optimal p_k. Thus, picking the better of the two points (if both are feasible), is a strategy that already allows rapid convergence towards the optimum, as we can expect the Hessian approximation to positive definite once the algorithm is close enough to a minimum.

A downside of this strategy is that it limits us to the poor steepest descent direction, when we are far away from the optimum. In this case, our steps are

likely constrained by the trust-region, even if B_k is positive definite. Thus, even though we have rapid convergence close to the optimum, it might take a very long time to get there. Our goal is therefore to slightly improve on this, without spending much more computation time.

Our idea is to slightly adapt the conjugate gradients algorithm, Algorithm 5 to the problem of finding the best point within the trust-region. The idea of the CG was that we picked an optimisation direction p_k that is *conjugate* to all previous directions and then obtained the optimal solution along the ray of this chosen search direction. We can still follow this general approach when adding the trust-region constraint, as for a chosen optimisation direction p_k it is still relatively easy to compute the optimal step in closed form. In fact, we did that already for a special case in the proof of Lemma 14. Choosing the step direction p_k would however be difficult as we already know that we cannot easily along the trust-region boundary. However, in our case the solver does not need to be able to find a critical point. As we only need to find any point that is at least as good as the one obtained by steepest-descent, we can just stop CG as soon as our solution is the boundary. As the first step of CG starts with the steepest descent direction, this always fulfills the requirement.

A Linear Algebra

Notation: We call $x \in \mathbb{R}^n$ a vector with elements $x_1, \ldots, x_n$. If we have several vectors $v_1, \ldots, v_k \in \mathbb{R}^n$, we denote their elements as $v_{i,j}$, $i = 1, \ldots, k$, $j = 1, \ldots, n$.

We define a $n \times m$ real-valued matrix A as $A \in \mathbb{R}^{n \times m}$ with elements A_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, m$. We call A_i the ith row of A. We denote the n dimensional identity matrix as I_n, that is $I_n \in \mathbb{R}^{n \times n}$ with $(I_n)_{ii} = 1$ and zero otherwise.

A.1 Basic definitions and Properties

Definition 10 (Linear (in)dependence)**.** *Let $x_1, \ldots, x_N \in \mathbb{R}^n$. We call a vector y linearly dependent on $x_1, \ldots, x_N$, if there exist scalars $w_1, \ldots, w_N \in \mathbb{R}$ such, that*

$$y = \sum_{i=1}^{N} w_i x_i \ .$$

Otherwise, we call y linearly independent of $x_1, \ldots, x_N$.

Definition 11 (Basis)**.** *We call a set of vectors $x_1, \ldots, x_n$ a basis of $\mathbb{R}^n$ if*

1. *The vectors $x_1, \ldots, x_n$ are linearly independent of all other elements in $x_1, \ldots, x_n$.*
2. *All vectors in $\mathbb{R}^n$ are linearly dependent on $x_1, \ldots, x_n$*

Definition 12 (Properties of the norm)**.** *Let $x, y \in \mathbb{R}^n$ and $a \in \mathbb{R}$. A norm $\|x\|$ fulfils*

1. $\|ax\| = |a|\|x\|$
2. $\|x\| = 0 \Rightarrow x = 0$
3. *(Triangle Inequality)* $\|x + y\| \leq \|x\| + \|y\|$

Definition 13 (Euclidean norm and inner product)**.** *Let $x, y \in \mathbb{R}^n$. We call*

$$x^T y = \sum_{i=1}^{n} x_i y_i$$

an inner- or scalar product on $\mathbb{R}^n$. The euclidean, or two-norm is defined as

$$\|x\| = \sqrt{x^T x}$$

Lemma 15 (Cauchy-Schwarz inequality)**.** *Let $x, y \in \mathbb{R}^n$. It holds*

$$|x^T y| \leq \|x\|\|y\| \ ,$$

with equality only when x and y are linearly dependent and $\|\cdot\|$ is the euclidean norm.

Lemma 16 (Positive Definiteness)**.** *Let $Q \in \mathbb{R}^{n \times n}$ be a symmetric matrix. We call Q positive semi-definite, if for all $x \in \mathbb{R}^n$ it holds*

$$x^T A x \geq 0 \ .$$

Moreover, if for all $x \in \mathbb{R}^n \setminus 0$ it holds

$$x^T A x > 0 \ ,$$

then we call A positive definite.

Lemma 17 (Eigenvalues and Eigenvector)**.** *Let $Q \in \mathbb{R}^{n \times n}$ be a matrix. We call $\lambda \in \mathbb{R}$ eigenvalue of Q with eigenvector $v \in \mathbb{R}^n \setminus \{0\}$ if it holds*

$$Qv = \lambda v \ .$$

Moreover, if Q is symmetric, then there exist n linearly independent eigenvectors $v_1, \ldots, v_n$ with eigenvalues $\lambda_1, \ldots \lambda_n$ such, that

$$Q = \sum_{i=1}^{n} \lambda_i v_i v_i^T \ .$$

In this case, let $V \in \mathbb{R}^{n \times n}$ be the matrix with ith column being v_i and Λ a diagonal matrix with $\Lambda_{ii} = \lambda_i$. We have

$$Q = V \Lambda V^T \ .$$

It further holds $V V^T = V^T V = I_n$.

B Calculus

We need a few basic results from analysis, which we will state shortly, without proof, but we give a short description of each:

We begin with the intermediate value theorem, that states that when we know the function value of a continuous function at the beginning and the end of an interval, then we also know that it will attain all function values in between on that interval.

Theorem 10 (Intermediate Value Theorem)**.** *Let $f : [a, b] \to \mathbb{R}$ be a continuous function. Further, let $l = \min\{f(a), f(b)\}$ and $u = \max\{f(a), f(b)\}$ be the rspective minimum and maximum values at the boundary of the interval $[a, b]$. Then, for each $y \in [l, u]$ there exists an $x \in [a, b]$ such, that*

$$f(x) = y$$

Another important theorem is the mean value theorem. It says, that if we know the value of a continuous differentiable function on an interval, then there exists a point in the interval where the derivative has the average slope.

Theorem 11 (Mean Value Theorem)**.** *Let $f : [a, b] \to \mathbb{R}$ be a continuous function that is differentiable on (a, b). Then, there exists a $x \in (a, b)$ such, that*

$$f'(x) = \frac{f(b) - f(a)}{b - a}$$

Next is the fundamental theorem of calculus, that relates Integrals and derivatives to each other.

Theorem 12 (Fundamental Theorem of Calculus)**.** *Let $f : [a, b] \to \mathbb{R}$ be a continuous function on (a, b). Let $F : [a, b] \to \mathbb{R}$ be the function defined as*

$$F(x) = \int_a^x f(t)\, dt \ .$$

Then, F is (uniformly) continuous on $[a, b]$ and differentiable on (a, b) and

$$F'(x) = f(x) \ .$$

Especially, it holds

$$\int_a^b f(x)\, dx = F(b) - F(a) \ .$$

The next theorem is an important theorem on function approximation. Taylor's theorem states that a k times differentiable function can be locally approximated well around a point x using its first k derivatives at x. The error of this approximation shrinks to zero the closer an evaluated point $y = a + x$ is to x and the error shrinks faster than all other terms in the approximation.

Theorem 13 (Taylor's theorem)**.** *Let $k \geq 1$ be an Integer and let $f : \mathbb{R} \to \mathbb{R}$ be k times differentiable at a point $x \in \mathbb{R}$. Then there exists a function $R(a)$ such, that*

$$f(x + a) = f(x) + f'(x)a + \frac{1}{2}f''(x)a^2 + \cdots + \frac{1}{k!}f^{(k)}(x)a^k + R(a) \ ,$$

where $f', \ldots, f^{(k)}$ are the first to kth derivative of f. It further holds

$$\lim_{a \to 0} \frac{R(a)}{a^k} = 0$$

In this course, we will mostly work with functions in several variables. For these functions we need to introduce proper generalisations of the derivative

Definition 14 (Partial derivative Gradient and Hessian)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable at a point $x \in \mathbb{R}$. We call the partial derivative of f with respect to x_i, $i = 1, \ldots, n$ the function*

$$\frac{\partial}{\partial x_i} f(x) = \lim_{t \to 0} \frac{f(x + te_i) - f(x)}{t} \ ,$$

where e_i is the ith standard basis vector, that is $e_{ii} = 1$ and zero otherwise.

We further define the gradient of f at x as the vector of partial derivatives

$$\nabla f(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix} .$$

If further, f is twice differentiable at x, we call the Hessian the matrix of second partial derivatives

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2}{\partial x_1 \partial x_1} f(x) & \dots & \frac{\partial^2}{\partial x_1 \partial x_n} f(x) \\ \vdots & \dots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} f(x) & \dots & \frac{\partial^2}{\partial x_n \partial x_n} f(x) \end{bmatrix} ,$$

where we use the shortcut

$$\frac{\partial^2}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i} \frac{\partial}{\partial x_j} = \frac{\partial}{\partial x_j} \frac{\partial}{\partial x_i}$$

With these definitions, a direct application of Taylor's Theorem to multiple variables can be obtained as follows. Let $f : \mathbb{R}^n \to \mathbb{R}$. Then we can pick two points $x, y \in \mathbb{R}^n$ and define a function $g(t) = f(x + t(y - x))$, which is the function value of f along the line that passes through points x and y. Applying Theorem 13 gives the following generalisation that we state for up to second derivative:

Theorem 14 (Taylor's theorem, multivariate)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ be differentiable at a point $x \in \mathbb{R}$ with gradient $\nabla f(x)$. Then there exists a function $R : \mathbb{R} \to \mathbb{R}$ such, that*

$$f(x + y) = f(x) + \nabla f(x)^T y + R(\|y\|) ,$$

with

$$\lim_{a \to 0} \frac{R(a)}{a} = 0$$

Further, if f is twice differentiable with Hessian $\nabla^2 f(x)$ then there exists a function $R : \mathbb{R} \to \mathbb{R}$ such, that

$$f(x + y) = f(x) + \nabla f(x)^T y + \frac{1}{2} y^T \nabla^2 f(x) y + R(\|y - x\|) ,$$

with

$$\lim_{a \to 0} \frac{R(a)}{a^2} = 0$$