MSE2122 - Nonlinear Optimization Lecture Notes V

Fernando Dias (based on previous version by Fabricio Oliveira)

September 28, 2023

Abstract

In this lecture, we present methods for solving unconstrained optimisation problems. We start by defining a general optimisation algorithm that will serve as a reference for deriving variants of optimisation methods. We concentrate first on how to generate step sizes using variants of unidimensional optimisation methods, the so called line searches. We also present the Armijo rule as an inexact line search method, widely used in state-of-the-art implementations of optimisation algorithms. Next, we focus on three variants of multidimensional methods, namely the coordinate descent (derivative free), the gradient descent method that relies in first order approximations, and the Newton's method, which relies on second-order information. We also discuss the effects of having exact and inexact line searches in each of these methods.

Contents

1 A prototype of an optimisation method

Most, if not all, optimisation methods are based on the conceptual notion of successively obtaining *directions* of potential improvement and suitable *step sizes* in this direction, until a convergence or termination criterion (collectively called stopping criteria) is satisfied.

Considering what we have seen so far, we have now the concepts required for describing several unconstrained optimisation methods. We start by posing a conceptual optimisation algorithm in a pseudocode structure. This will be helpful in identifying the elements that differentiate the methods we will discuss.

Algorithm 1 Conceptual optimisation algorithm

```
1: initialise. iteration count $k = 0$, starting point $x_0$
2: while stopping criteria are not met do
3:     compute direction $d_k$
4:     compute step size $\lambda_k$
5:     $x_{k+1} = x_k + \lambda_k d_k$
6:     $k = k + 1$
7: end while
8: return $x_k$.
```

Algorithm 1 has two main elements, namely the computation of the direction d_k and the step size λ_k at each iteration k. In what follows, we present some univariate optimisation methods that can be employed to calculate step sizes λ_k. These methods are commonly referred to as *line search methods.*

2 Line search methods

Finding an optimal step size λ_k is in itself an optimisation problem. The name line search refers to the fact that it consists of a unidimensional search as $\lambda_k \in \mathbb{R}$.

Suppose that $f : \mathbb{R}^n \mapsto \mathbb{R}$ is differentiable. We define the unidimensional function $\theta : \mathbb{R} \mapsto \mathbb{R}$ as:

$$\theta(\lambda) = f(x + \lambda d).$$

Assuming differentiability, we can use the first-order necessary condition $\theta'(\lambda) = 0$ to obtain optimal values for the step size λ. This means solving the system:

$$\theta'(\lambda) = d^\top \nabla f(x + \lambda d) = 0$$

which might pose challenges. First, $d^\top \nabla f(x + \lambda d)$ is often nonlinear in λ, with optimal solutions not trivially resting at boundary points for an explicit domain of λ. Moreover, recall that $\theta'(\lambda) = 0$ is not a sufficient condition for optimality in general, unless properties such as convexity can be inferred.

In what follows, we assume that strict quasiconvexity holds and therefore $\theta'(\lambda) = 0$ becomes necessary and sufficient for optimality. In some contexts, unidimensional strictly quasiconvex functions are called *unimodal.*

Theorem 2.1 establishes the mechanism underpinning line search methods. In that, we use the assumption that the function has a unique minimum (a consequence of being strictly quasiconvex) to successively reduce the search space until the optimal is contained in a sufficiently small interval l within an acceptable tolerance.

Theorem 2.1. Line search reduction

Let $\theta : \mathbb{R} \to \mathbb{R}$ be strictly quasiconvex over the interval $[a, b]$, and let $\lambda, \mu \in [a, b]$ such that $\lambda < \mu$. If $\theta(\lambda) > \theta(\mu)$, then $\theta(z) \geq \theta(\mu)$ for all $z \in [a, \lambda]$. If $\theta(\lambda) \leq \theta(\mu)$, then $\theta(z) \geq \theta(\lambda)$ for all $z \in [\mu, b]$.

Figure 1 provides an illustration of Theorem 2.1. The line below the x-axis illustrates how the search

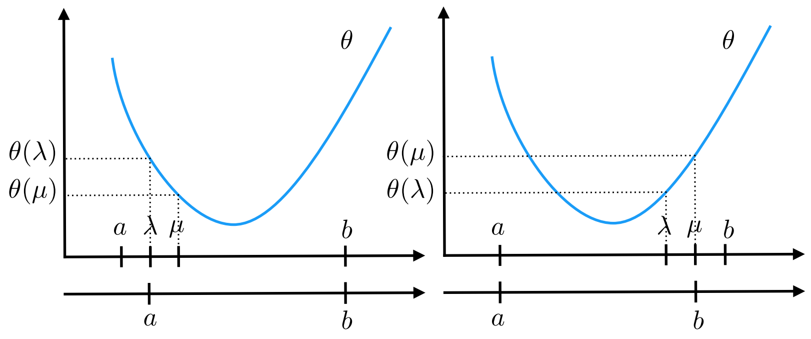

Figure 1: Applying Theorem 2.1 allows to iteratively reduce the search space.

space can be reduced between two successive iterations. In fact, most line search methods will iteratively reduce the search interval (represented by $[a, b]$) until the interval is sufficiently small to be considered "a point" (i.e., is smaller than a set threshold l).

Line searches are *exact* when optimal step sizes λ_k^* are calculated at each iteration k, and inexact when arbitrarily good approximations for λ_k^* are used instead. As we will see, there is a trade-off between the number iterations required for convergence and the time taken per iteration that must be taken into account when choosing between exact and inexact line searches.

2.1 Exact line searches

Exact methods are designed to return the optimal step value λ^* within a pre-specified tolerance l. In practice, it means that these methods return an interval $[a_k, b_k]$ such that $b_k - a_k \leq l$.

2.1.1 Uniform search

The uniform search consists of breaking the search domain $[a, b]$ into N slices of uniform size $\delta = \frac{|b-a|}{N}$. This leads to a one-dimensional grid with grid points $a_n = a_0 + n\delta, n = 0 \dots N$ where $a_0 = a$ and $a_N = b$. We can then set $\hat{\lambda}$ to be:

$$\hat{\lambda} = arg \min_{i=0,\dots,n} f(a_i)$$

From Theorem 2.1, we know that the optimal step size $\lambda^* \in [\hat{\lambda} - \delta, \hat{\lambda} + \delta]$. The process can then be repeated, by making $a = \hat{\lambda} - \delta$ and $b = \hat{\lambda} + \delta$ (see Figure 2). until $|a - b|$ is less than a prespecified tolerance l. Without enough repetition of the search, the uniform search becomes an inexact search.

This type of search is particularly useful when setting values for hyperparameters in algorithms (that is, user defined parameters that influence the behaviour of the algorithm) of performing any sort of search in a grid structure. One concept related to this type of search is what is known as the *coarse-to-fine approach*. Coarse-to-fine approaches use sequences of increasingly fine approximations (i.e., gradually increasing n) to obtain computational savings in terms of function evaluations. In fact, the number of function evaluations a line search method executes is one of the indicators of its efficiency.

2.1.2 Dichotomous search

The *dichotomous search* is an example of a sequential line search method, in which evaluations of the function θ at a current iteration k are reused in the next iteration $k + 1$ to minimise the number of function evaluations and thus improve performance.

The word dichotomous refer to the mutually exclusive parts that the search interval $[a, b]$ is divided at each iteration. We start by defining a distance margin ϵ and defining two reference points $\lambda = \frac{a+b}{2} - \epsilon$ and $\mu = \frac{a+b}{2} + \epsilon$. Using the function values $\theta(\lambda)$ and $\theta(\mu)$, we proceed as follows.

1. If $\theta(\lambda) < \theta(\mu)$, then *move to the left* by making $a_{k+1} = a_k$ and $b_{k+1} = \mu_k$;
2. Otherwise, if $\theta(\lambda) > \theta(\mu)$, then *move to the right* by making $a_{k+1} = \lambda_k$ and $b_{k+1} = b_k$.

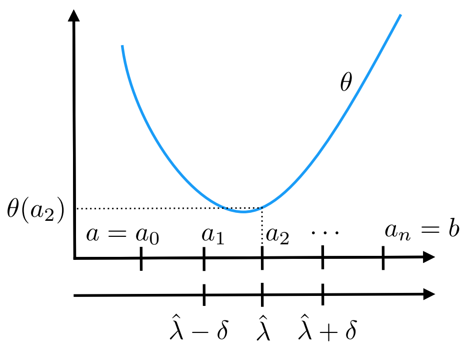

Figure 2: Grid search with 5 points; Note that $\theta(a_2) = \min_{i=0,\ldots,n} \theta(a_i)$.

Notice that, the assumption of strict quasiconvexity implies that $\theta(\lambda) = \theta(\mu)$ cannot occur, but in a more general setting one must make sure a criterion for resolving the tie. Once the new search interval $[a_{k+1}, b_{k+1}]$ is updated, new reference points λ_{k+1} and μ_{k+1} are calculated and the process is repeated until $|a - b| \leq l$. The method is summarised in Algorithm 2. Notice that, at any given iteration k, one can calculate what will be the size $|a_{k+1} - b_{k+1}|$, given by:

$$b_{k+1} - a_{k+1} = \frac{1}{2^k}(b_0 - a_0) + 2\epsilon\left(1 - \frac{1}{2^k}\right).$$

This is useful in that it allows predicting the number of iterations Algorithm 2 will require before convergence. Figure 3 illustrates the process for two distinct functions. Notice that the employment of the central point $\frac{a+b}{2}$ as the reference to define the points λ and μ turns the method robust in terms of interval reduction at each iteration.

Algorithm 2 Dichotomous search

```
1:  initialise. distance margin $\epsilon > 0$, tolerance $l > 0$, $[a_0, b_0] = [a, b]$, $k = 0$
2:  while $b_k - a_k > l$ do
3:      $\lambda_k = \frac{a_k+b_k}{2} - \epsilon$, $\mu_k = \frac{a_k+b_k}{2} + \epsilon$
4:      if $\theta(\lambda_k) < \theta(\mu_k)$ then
5:          $a_{k+1} = a_k$, $b_{k+1} = \mu_k$
6:      else
7:          $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$
8:      end if
9:      $k = k + 1$
10: end while
11: return $\overline{\lambda} = \frac{a_k+b_k}{2}$
```

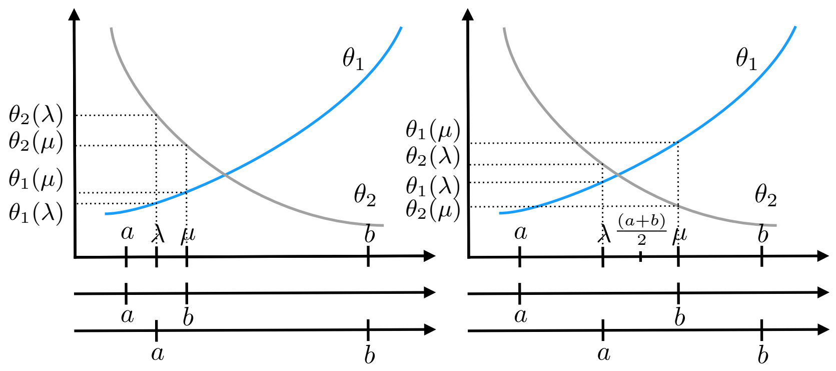

Figure 3: Using the midpoint $(a + b)/2$ and Theorem 2.1 to reduce the search space.

2.1.3 Golden section search*

The *golden section search* is named after the *golden ratio* $\varphi = \frac{1+\sqrt{5}}{2}$, of which the inverse is used as the ratio of reduction for the search interval $[a, b]$ at each iteration.

Consider that, once again, we rely on two reference points λ_k and μ_k. The method is a consequence of imposing two requirements for the line search:

1. the reduction in the search interval should not depend on whether $\theta(\lambda_k) > \theta(\mu_k)$ or vice-versa.

2. at each iteration, we perform a single function evaluation, thus making $\lambda_{k+1} = \mu_k$ if $\theta(\lambda_k) > \theta(\mu_k)$ or vice-versa.

From requirement 1, we can infer that $b_{k+1} - a_{k+1} = b_k - \lambda_k = \mu_k - a_k$ is required. To find the interval reduction rate $\alpha \in (0, 1)$ that would allow so, we define $\mu_k = a_k + \alpha(b_k - a_k)$ and, consequently, $\lambda_k = a_k + (1 - \alpha)(b_k - a_k)$. Notice that this makes $b_{k+1} - a_{k+1} = \alpha(b_k - a_k)$.

Notice the following. Suppose that $\theta(\lambda_k) > \theta(\mu_k)$ at iteration k. We then make $a_{k+1} = \lambda_k$ and $b_{k+1} = b_k$, a "movement to the right". From requirement 2, we also make $\lambda_{k+1} = \mu_k$ so that $\theta(\lambda_{k+1}) = \theta(\mu_k)$, avoiding a function evaluation.

From the above, we can calculate the ratio α that would allow the method to work. Notice that:

$$\begin{aligned}
&\lambda_{k+1} = \mu_k \\
&a_{k+1} + (1 - \alpha)(b_{k+1} - a_{k+1}) = \mu_k \\
&(1 - \alpha)[\alpha(b_k - a_k)] = \mu_k - \lambda_k \\
&(\alpha - \alpha^2)(b_k - a_k) = a_k + \alpha(b_k - a_k) - [a_k + (1 - \alpha)(b_k - \alpha_k)] \\
&\alpha^2 + \alpha - 1 = 0
\end{aligned}$$

to which $\alpha = \frac{2}{1+\sqrt{5}} = 0.618... = \frac{1}{\varphi}$ is the positive solution. Clearly, the same result is obtained if one consider $\theta(\lambda_k) < \theta(\mu_k)$. Algorithm 3 summarises the golden section search. Notice that at each iteration, only a single additional function evaluation is required.

Algorithm 3 Golden section search

```
1: initialise. tolerance $l > 0$, $[a_0, b_0] = [a, b]$, $\alpha = 0.618$, $k = 0$
2: $\lambda_k = a_k + (1 - \alpha)(b_k - a_k)$, $\mu_k = a_k + \alpha(b_k - a_k)$
3: while $b_k - a_k > l$ do
4:     if $\theta(\lambda_k) > \theta(\mu_k)$ then
5:         $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$, $\lambda_{k+1} = \mu_k$, and
6:         $\mu_{k+1} = a_{k+1} + \alpha(b_{k+1} - a_{k+1})$. Calculate $\theta(\mu_{k+1})$
7:     else
8:         $a_{k+1} = a_k$, $b_{k+1} = \mu_k$, $\mu_{k+1} = \lambda_k$, and
9:         $\lambda_{k+1} = a_{k+1} + (1 - \alpha)(b_{k+1} - a_{k+1})$. Calculate $\theta(\lambda_{k+1})$
10:    end if
11:    $k \leftarrow k + 1$
12: end while
13: return $\overline{\lambda} = \frac{a_k + b_k}{2}$
```

Comparing the above method for a given accuracy l, the required number of function evaluations is:

$$\min \left\{ n : \begin{array}{l} \text{uniform: } n \geq \frac{b_1 - a_1}{l/2} - 1 \\ \text{dichotomous: } (1/2)^{n/2} \leq \frac{l}{b_1 - a_1} \\ \text{golden section: } (0.618)^{n-1} \leq \frac{l}{b_1 - a_1} \end{array} \right\}$$

For example: suppose we set $[a, b] = [-10, 10]$ and $l = 10^{-6}$. Then the number of iterations required for convergence is

- uniform: $n = 4 \times 10^6$;

- dichotomous: $n = 49$;

- golden section: $n = 36$.

A variant of the golden section method uses Fibonacci numbers to define the ratio of interval reduction. Despite being marginally more efficient in terms of function evaluations, the overhead of calculating Fibonacci numbers has to be taken into account.

2.1.4 Bisection search

Differently form the previous methods, the bisection search relies on derivative information to infer whether how the search interval should be reduced. For that, we assume that $\theta(\lambda)$ is differentiable and convex.

We proceed as follows. If $\theta'(\lambda_k) = 0$, then λ_k is a minimiser. Otherwise:

1. if $\theta'(\lambda_k) > 0$, then, for $\lambda > \lambda_k$, we have $\theta'(\lambda_k)(\lambda - \lambda_k) > 0$, which implies $\theta(\lambda) \geq \theta(\lambda_k)$ since θ is convex. Therefore, the new search interval becomes $[a_{k+1}, b_{k+1}] = [a_k, \lambda_k]$.
2. if $\theta'(\lambda_k) < 0$, we have $\theta'(\lambda_k)(\lambda - \lambda_k) > 0$ (and thus $\theta(\lambda) \geq \theta(\lambda_k)$) for $\lambda < \lambda_k$. Thus, the new search interval becomes $[a_{k+1}, b_{k+1}] = [\lambda_k, b_k]$.

As in the dichotomous search, we set $\lambda_k = \frac{1}{2}(b_k + a_k)$, which provides robust guarantees of search interval reduction. Notice that the dichotomous search can be seen as a bisection search in which the derivative information is estimated using the difference of function evaluation at two distinct points. Algorithm 4 summarises the bisection method.

Algorithm 4 Bisection method

```
1: initialise. tolerance $l > 0$, $[a_0, b_0] = [a, b]$, $k = 0$
2: while $b_k - a_k > l$ do
3:     $\lambda_k = \frac{(b_k + a_k)}{2}$ and evaluate $\theta'(\lambda_k)$
4:     if $\theta'(\lambda_k) = 0$ then return $\lambda_k$
5:     else if $\theta'(\lambda_k) > 0$ then
6:         $a_{k+1} = a_k$, $b_{k+1} = \lambda_k$
7:     else
8:         $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$
9:     end if
10:    $k \leftarrow k + 1$
11: end while
12: return $\overline{\lambda} = \frac{a_k + b_k}{2}$
```

2.2 Inexact line search

Often, it is worth sacrificing optimality of the step size λ^k for the overall efficiency of the solution method in terms of solution time.

There are several heuristics that can be employed to define step sizes and their performance are related to how the directions d_k are defined in Algorithm 1. Next, we present the *Armijo rule*, arguably the most used technique to obtain step sizes in efficient implementations of optimisation methods.

2.2.1 Armijo rule

The Armijo rule is a condition that is tested to decide whether a current step size $\overline{\lambda}$ is acceptable or not. The step size $\overline{\lambda}$ is considered acceptable if:

$$f(x + d\overline{\lambda}) - f(x) \leq \alpha\overline{\lambda}\nabla f(x)^\top d.$$

One way of understanding the Armijo rule is to look at what it means in terms of the function $\theta(\lambda) = f(x + \lambda d)$. Notice that, at $\lambda = 0$, the Armijo rule becomes:

$$\begin{aligned} \theta(\overline{\lambda}) - \theta(0) &\leq \alpha\overline{\lambda}\theta'(0) \\ \theta(\overline{\lambda}) &\leq \theta(0) + \alpha\overline{\lambda}\theta'(0). \quad (1) \\ & \quad (2) \end{aligned}$$

That is, $\theta(\overline{\lambda})$ has to be less than the deflected linear extrapolation of θ at $\lambda = 0$. The deflection is given by the pre-specified parameter α. In case $\overline{\lambda}$ does not satisfy the test in (1), $\overline{\lambda}$ is reduced by a factor $\beta \in (0,1)$ until the test in (1) is satisfied.

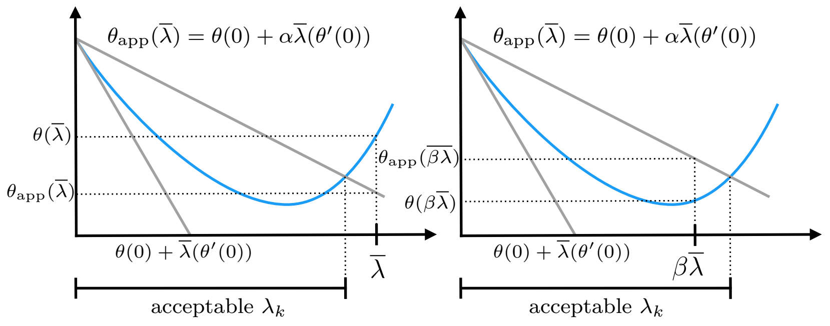
$\theta_{\text{app}}(\overline{\lambda}) = \theta(0) + \alpha\overline{\lambda}(\theta'(0))$
$\theta(\overline{\lambda})$
$\theta_{\text{app}}(\overline{\lambda})$
$\theta(0) + \overline{\lambda}(\theta'(0))$
$\overline{\lambda}$
acceptable λ_k
$\theta_{\text{app}}(\overline{\lambda}) = \theta(0) + \alpha\overline{\lambda}(\theta'(0))$
$\theta_{\text{app}}(\beta\overline{\lambda})$
$\theta(\beta\overline{\lambda})$
$\theta(0) + \overline{\lambda}(\theta'(0))$
$\beta\overline{\lambda}$
acceptable λ_k

Figure 4: At first $\lambda_0 = \overline{\lambda}$ is not acceptable; after reducing the step size to $\lambda_1 = \beta\overline{\lambda}$, it enters the acceptable range where $\theta(\lambda_k) \leq \theta_{\text{app}}(\lambda_k) = \theta(0) + \alpha\lambda_k(\theta'(0))$.

In Figure 4, we can see the acceptable region for the Armijo test. At first, $\overline{\lambda}$ does not satisfy the condition (1), being then reduced to $\beta\overline{\lambda}$, which, in turn, satisfies (1). In this case, λ_k would have been set $\beta\overline{\lambda}$. Suitable values for α are within $(0, 0.5]$ and for β are within $(0,1)$, trading of precision (higher values) and number of tests before acceptance (lower values).

The Armijo rule is called *backtracking* in some contexts, due to the successive reduction of the step size caused by the factor $\beta \in (0,1)$. Some variants might also include rules that prevent the step size from becoming too small, such as $\theta(\delta\overline{\lambda}) \geq \theta(0) + \alpha\delta\overline{\lambda}\theta'(0)$, with $\delta > 1$.

3 Unconstrained optimisation methods

We now focus on developing methods that can be employed to optimise $f : \mathbb{R}^n \mapsto \mathbb{R}$. We start with coordinate descent method, which is derivative free, to then discuss the gradient method and Newton's method. In essence, the main difference between the three methods is how the directions d_k in Algorithm 1 are determined. Also, all of these methods rely on line searches to define optimal step sizes, which can be any of the methods seen before or any other unidimensional optimisation method.

3.1 Coordinate descent

The *cordinate descent method* relies on a simple yet powerful idea. By focusing on one coordinate at the time, the method trivially derives directions d having $d_i = 1$ for coordinate i and $d_{j \neq i} = 0$ otherwise. As one would suspect, the order in which the coordinates are selected influences the performance of the algorithm. Some known variants include:

1. **Cyclic:** coordinates are considered in order $1, \dots, n$;
2. **Double-sweep:** swap the coordinate order at each iteration;
3. **Gauss-Southwell:** choose components with largest $\frac{\partial f(x)}{\partial x_i}$;
4. **Stochastic:** coordinates are selected at random.

Algorithm 5 summarises the general structure of the coordinate descent method. Notice that the for-loop starting in Line 3 uses the cyclic variant of the coordinate descent method.

Figure 5 shows the progress of the algorithm when applied to solve

$$f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$$

using the golden section method as line search.

The coordinate descent method is the strategy employed in several other methods, such as the *Gauss-*

Algorithm 5 Coordinate descent method (cyclic)

```
1: initialise. tolerance $\epsilon > 0$, initial point $x^0$, iteration count $k = 0$
2: while $||x^{k+1} - x^k|| > \epsilon$ do
3:     for $j = 1, \dots n$ do
4:         $d = \{d_i = 1,\ \text{if } i = j; d_i = 0,\ \text{if } i \neq j\}$
5:         $\overline{\lambda}_j = \text{argmin}_{\lambda \in \mathbb{R}}\{f(x_j^k + \lambda d_j)\}$
6:         $x_j^{k+1} = x_j^k + \overline{\lambda}_j d_j$
7:     end for
8:     $k = k + 1$
9: end while
10: return $x^k$
```

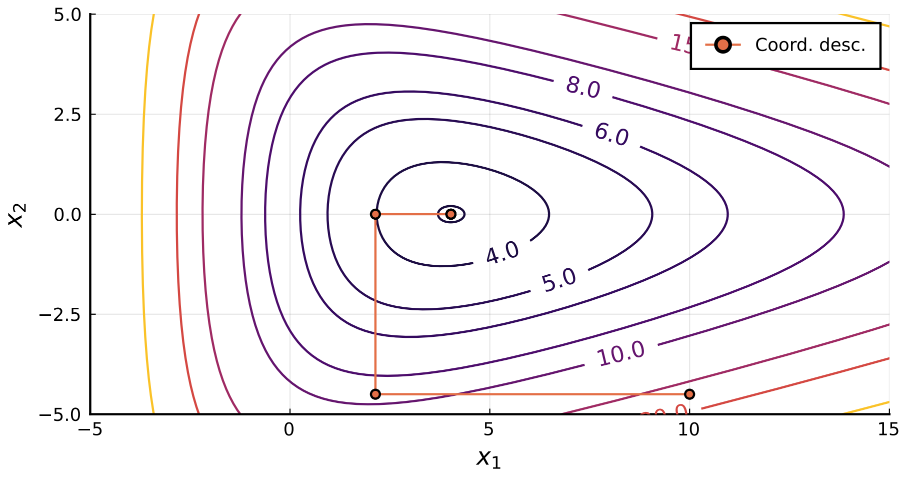

Figure 5: Coordinate descent method applied to f. Convergence is observed in 4 steps for a tolerance $\epsilon = 10^{-5}$

Seidel method for solving linear system of equations, which is why some references refer to each of these iterations as Gauss-Seidel steps. Also, when a collection of coordinates is used to derive a direction, the term *block coordinate descent* is used, though a method for deriving directions for each block is still necessary, for example the gradient method presented next.

3.2 Gradient (descent) method

The *gradient descent* uses the function gradients as the search direction d. Before we present the method, let us present a result that justifies the use of gradients to derive search directions.

Lemma 3.1. S ppose that $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable at $x \in \mathbb{R}^n$ and $\nabla f(x) \neq 0$. Then $\overline{d} = -\frac{\nabla f(x)}{||\nabla f(x)||}$ is the direction of steepest descent of f at x.

Proof. From differentiability of f, we have:

$$f'(x; d) = \lim_{\lambda \to 0^+} \frac{f(x + \lambda d) - f(x)}{\lambda} = \nabla f(x)^\top d.$$

Thus, $\overline{d} = \text{argmin}_{||d|| \leq 1}\{\nabla f(x)^\top d\} = -\frac{\nabla f(x)}{||\nabla f(x)||}$

😎

In the proof, we use the differentiability to define a directional derivative for f at direction d, that is, the change in the value of f by a move of size $\lambda > 0$ in the direction d, which is given by $\nabla f(x)^\top d$. If we minimise this term in d for $||d||_2 \leq 1$, we observe that d is a vector of length one that has the opposite direction of $\nabla f(x)$, thus $d = -\frac{\nabla f(\overline{x})}{||\nabla f(\overline{x})||}$.

That provides us with the insight that we can use $\nabla f(\overline{x})$ to derive (potentially good) directions for optimising f. Notice that the direction employed is the opposite direction of the gradient for minimisation problems, being the opposite in case of maximisation. That is the reason why the gradient method is called the *steepest descent* method in some references, though gradient and steepest descent might refer to different methods in specific contexts.

Using the gradient $\nabla f(\overline{x})$ is also a convenience as it allows for the definition of a straightforward convergence condition. Notice that, if $\nabla f(\overline{x}) = 0$, then the algorithm stalls, as $x_{k+1} = x_k + \lambda_k d_k = x_k$. In other words, the algorithm converges to points $x \in \mathbb{R}^n$ that satisfy the first-order necessary conditions $\nabla f(\overline{x}) = 0$.

The gradient method has many known variants that try to mitigate issues associated with the poor convergence caused by the natural 'zigzagging' behaviour of the algorithm (see, for example the gradient method *with momentum* and the *Nesterov* method).

There are also variants that only consider the partial derivatives of some (and not all) of the dimensions $i = 1, \dots, n$ forming *blocks* of coordinates at each iteration. If these blocks are randomly formed, these methods are known as *stochastic gradient* methods.

In Algorithm 6 we provide a pseudocode for the gradient method. In Line 2, the stopping condition for the while-loop is equivalent of testing $\nabla f(\overline{x}) = 0$ for a tolerance ϵ.

Algorithm 6 Gradient method

```
1: initialise. tolerance $\epsilon > 0$, initial point $x_0$, iteration count $k = 0$.
2: while $||\nabla f(x_k)|| > \epsilon$ do
3:     $d = -\frac{\nabla f(x_k)}{||\nabla f(\overline{x})||}$
4:     $\overline{\lambda} = \text{argmin}_{\lambda \in \mathbb{R}}\{f(x_k + \lambda d)\}$
5:     $x_{k+1} = x_k + \overline{\lambda} d_j$
6:     $k = k + 1$
7: end while
8: return $x_k$.
```

Figure 6 presents the progress of the gradient method using exact (bisection) and inexact (Armijo rule with $\alpha = 0.1$ and $\beta = 0.7$) line searches. As can be expected, when an inexact line search is employed, the method overshoots slightly some of the steps, taking a few more iterations to converge.

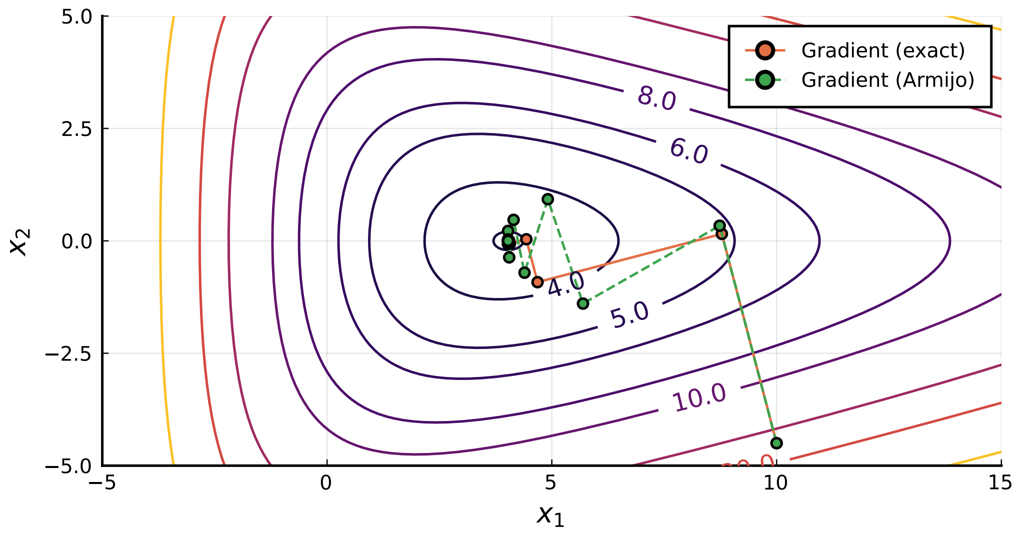

Figure 6: Gradient method applied to f. Convergence is observed in 10 steps using exact line search and 19 using Armijo's rule (for $\epsilon = 10^{-5}$)

3.3 Newton's method

One can think of gradient methods as using first-order information to derive directions of improvement, while *Newton's method* consists of a step forward also incorporating second-order information.

This can be shown to produce better convergence properties, but at the expense of the extra computational burden incurred by calculating and manipulating Hessian matrices.

The idea of the Newton's method is the following. Consider the second-order approximation of f at x_k, which is given by:

$$q(x) = f(x_k) + \nabla f(x_k)^\top (x - x_k) + \frac{1}{2}(x - x_k)^\top H(x_k)(x - x_k)$$

The method uses as direction d that of the extremum of the quadratic approximation at x_k, which can be obtained from the first-order condition $\nabla q(x) = 0$. This renders:

$$\nabla q(x) = \nabla f(x_k) + H(x_k)(x - x_k) = 0. \tag{3}$$

(4)

Assuming that $H^{-1}(x_k)$ exists, we can use (3) to obtain the following update rule, which is known as the *Newton step*:

$$x_{k+1} = x_k - H^{-1}(x_k)\nabla f(x_k) \tag{5}$$

(6)

Notice that the "pure" Newton's method has embedded in the direction of the step, its length (i.e., the step size) as well. In practice, the method uses $d = -H^{-1}(x_k)\nabla f(x_k)$ as a direction combined with a line search to obtain optimal step sizes and prevent divergence (that is, converge to $-\infty$) in cases where the second-order approximation might lead to divergence. Fixing $\lambda = 1$ renders the natural Newton's method, as derived in (5). The Newton's method can also be seen as employing Newton-Raphson method to solve the system of equations that describe the first order conditions of the quadratic approximation at x_k.

Figure 7 shows the calculation of direction $d = -H^{-1}(x_k)\nabla f(x_k)$ for the first iteration of the Newton's method. Notice that the direction is the same as the that of the minimum of the quadratic approximation $q(x)$ at x_k. The employment of a line search allows for overshooting the exact minimum, making the search more efficient.

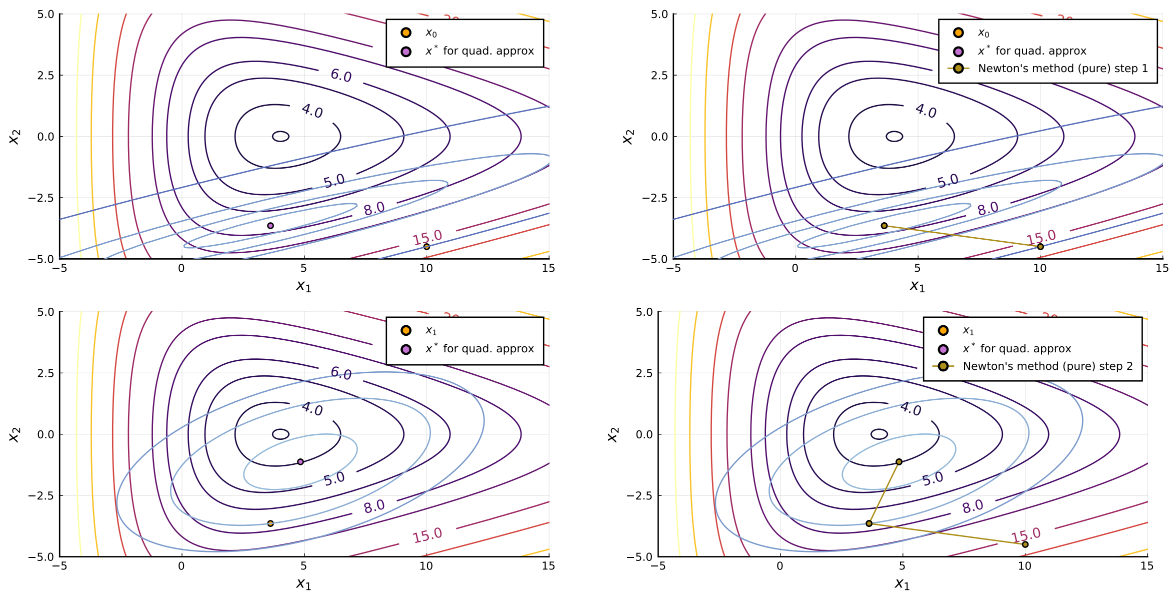

Figure 7: The calculation of the direction $d = x^* - x_0$ in the first two iterations of the Newton's method with step size λ fixed to 1 (the pure Newton's method, in left to right, top to bottom order). Notice in blue the level curves of the quadratic approximation of the function at the current point x_k and how it improves from one iteration to the next.

The Newton's method might diverge if the initial point is too far from the optimal and fixed step sizes (such as $\lambda = 1$) are used, since the quadratic approximation minimum and the actual function minimum can become drastically and increasingly disparate. Levenberg-Marquardt method and other trust-region-based variants address convergence issues of the Newton's method. As a general rule, combining the method with an exact line search of a criteria for step-size acceptance that require improvement (such as employing the Armijo rule for defining the step sizes) if often sufficient for guaranteed convergence. Figure 8 compares the convergence of the pure Newton's method and the method employing an exact line search.

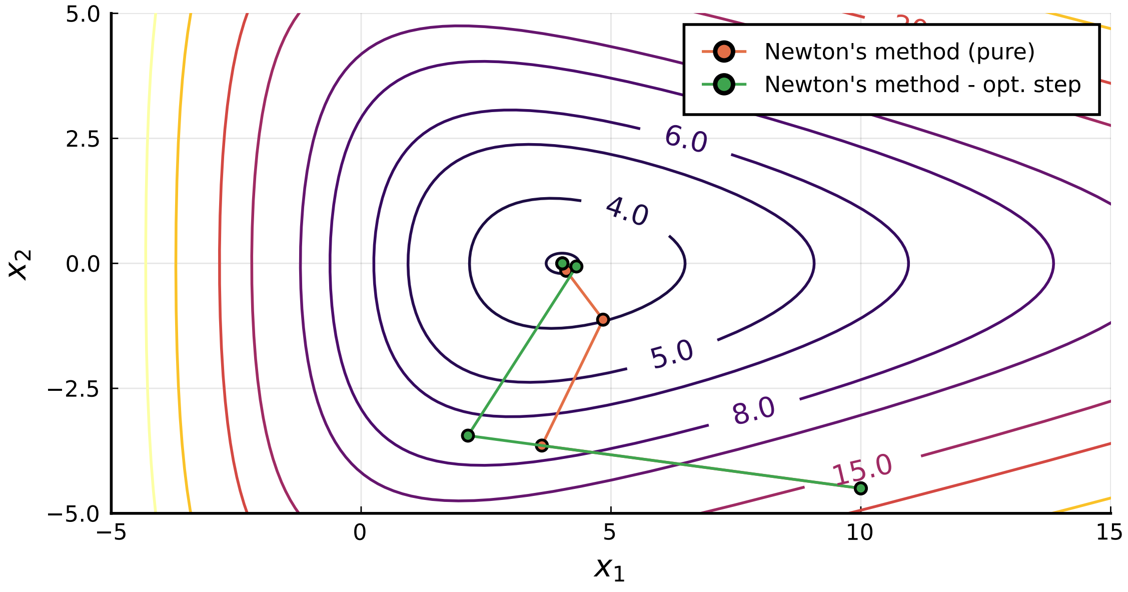

Figure 8: A comparison of the trajectory of both Newton's method variants. Notice that in the method using the exact line search, wile the direction $d = x^* - x_0$ is utilised, the step size is larger in the first iteration.

Algorithm 7 presents a pseudocode for the Newton's method. Notice that in Line 3, an inversion operation is required. One might be cautious about this operation, since as $\nabla f(x_k)$ tends to zero, the Hessian $H(x_k)$ tends to become singular, potentially causing numerical instabilities.

Algorithm 7 Newton's method

```
1: **initialise.** tolerance $\epsilon > 0$, initial point $x_0$, iteration count $k = 0$.
2: **while** $||\nabla f(x_k)|| > \epsilon$ **do**
3:     $d = -H^{-1}(x_k)\nabla f(x_k)$
4:     $\overline{\lambda} = \text{argmin}_{\lambda \in \mathbb{R}}\{f(x_k + \lambda d)\}$
5:     $x_{k+1} = x_k + \overline{\lambda} d$
6:     $k = k + 1$
7: **end while**
8: **return** $x_k$
```

Figure 9 shows the progression of the Newton's method for f with exact and inexact line searches.

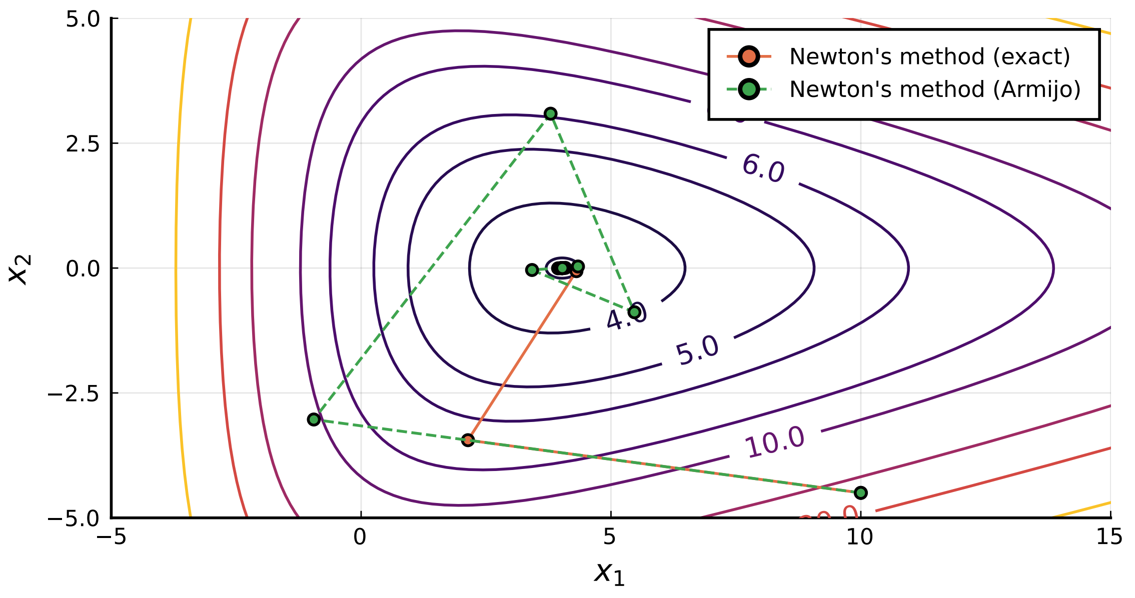

Figure 9: Newton's method applied to f. Convergence is observed in 4 steps using exact line search and 27 using Armijo's rule ($\epsilon = 10^{-5}$)