Lecture 6 - Unconstrained optimisation methods II

Fabricio Oliveira

Last update: October 18, 2021

Abstract. We discuss two additional unconstrained optimisation methods that builds upon more sophisticated ideas to encode curvature information when obtaining search directions. The first is the conjugate gradient method, that uses the notion of conjugate directions to devise search directions. The second is a class of methods known as quasi-Newton methods, which rely on approximations for the Hessian matrix that preclude the need of calculating inverses and second-order derivatives. We also discuss the important concepts of complexity, convergence, and conditioning, illustrating how these can influence the performance of the optimisation method.

Outline of this lecture

1 Unconstrained optimisation methods

We will now discuss to variants of the gradient and Newton methods that try to exploit the computational simplicity of gradient methods while encoding of curvature information as the Newton's method, but without explicitly relying on second-order derivatives (i.e., Hessian matrices).

1.1 Conjugate gradient method

The conjugate gradient method use the notion of *conjugacy* to guide the search for optimal solutions. The original motivation for the method comes from quadratic problems, in which one can use conjugacy to separate the search for the optimum of $f : \mathbb{R}^n \mapsto \mathbb{R}$ into n exact steps.

1.1.1 The concept of conjugacy

Let us first define the concept of conjugacy.

Definition 1. *Let H be an $n \times n$ symmetric matrix. The vectors $d_1, \dots, d_n$ are called (H-)conjugate if they are linearly independent and $d_i^\top H d_j = 0$, for all $i, j = 1, \dots, n$ such that $i \neq j$.*

Notice that H-conjugacy (or simply conjugacy) is a generalisation of orthogonality under the linear transformation imposed by the matrix H. Notice that orthogonal vectors are H-conjugate for $H = I$. Figure 1 illustrate the notion of conjugacy between two vectors d_1 and d_2 that are H-conjugate, being H the Hessian of the underlying quadratic function. Notice how it allows one to generate, from direction d_1, a direction d_2 that, if used in combination with an exact line search, would take us to the centre of the curve.

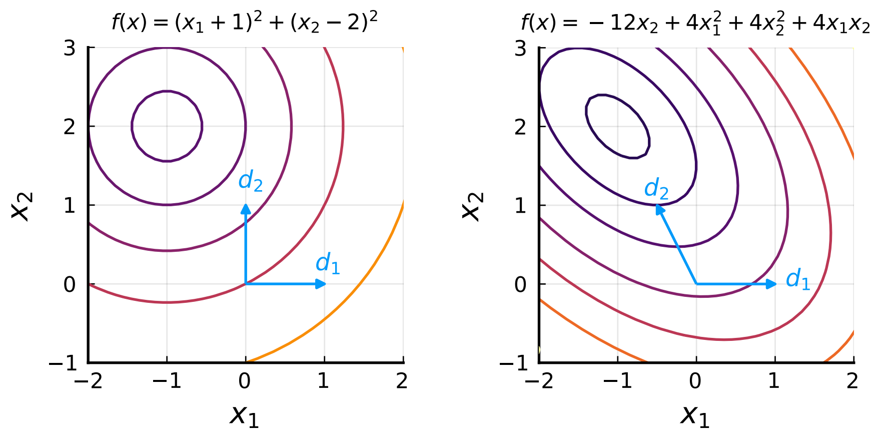

Figure 1: d_1 and d_2 are H-conjugates; on the left, $H = I$.

One can use H-conjugate directions to find optimal solutions for the quadratic function $f(x) = c^\top x + \frac{1}{2}x^\top Hx$, where H is a symmetric matrix. Suppose we know directions $d_1, \dots, d_n$ that are H-conjugate. Then, given an initial point x_0, any point x can be described as $x = x_0 + \sum_{j=1}^{n} \lambda_j d_j$. We can then reformulate $f(x)$ as a function of the step size λ, i.e.,

$$\begin{aligned} f(x) = F(\lambda) &= c^\top (x_0 + \sum_{j=1}^{n} \lambda_j d_j) + \frac{1}{2}(x_0 + \sum_{j=1}^{n} \lambda_j d_j)^\top H(x_0 + \sum_{j=1}^{n} \lambda_j d_j) \\ &= \sum_{j=1}^{n} \left[c^\top (x_0 + \lambda_j d_j) + \frac{1}{2}(x_0 + \lambda_j d_j)^\top H(x_0 + \lambda_j d_j)\right]. \end{aligned}$$

This reformulation exposes an important properties that having conjugate directions $d_1, \dots, d_n$ allows us to explore: separability. Notice that $F(\lambda) = \sum_{j=1}^{n} F_j(\lambda_j)$, where $F_j(\lambda_j)$ is given by

$$F_j(\lambda_j) = c^\top (x_0 + \lambda_j d_j) + \frac{1}{2}(x_0 + \lambda_j d_j)^\top H(x_0 + \lambda_j d_j),$$

and is, ultimately, a consequence of the linear independence of the conjugate directions. Assuming that H is positive definite, and thus that first-order conditions are necessary and sufficient for optimality, we can then calculate optimal $\overline{\lambda}_j$ for $j = 1, \dots, n$ as

$$\begin{aligned} F_j'(\lambda_j) &= 0 \\ c^\top d_j + x_0^\top H d_j + \lambda_j d_j^\top H d_j &= 0 \\ \overline{\lambda}_j &= -\frac{c^\top d_j + x_0^\top H d_j}{d_j^\top H d_j}, \text{ for all } j = 1, \dots, n. \end{aligned}$$

This result can be used to devise an iterative method that can obtain optimal solution for quadratic functions in exactly n iterations. From an initial point x_0 and a collection of H-conjugate directions $d_1, \dots, d_n$, the method consists of the successively executing the following step

$$x_k = x_{k-1} + \lambda_k d_k, \text{ where } \lambda_k = -\frac{c^\top d_k + x_{k-1}^\top H d_k}{d_k^\top H d_k}$$

Notice the resemblance this method hold with the coordinate descent method. In case $H = I$, then the coordinate directions given by $d_i = 1$ and $d_{j \neq i} = 0$ are H-conjugate and thus, the coordinate descent method converges in two iterations. Figure 2 illustrates this behaviour. Notice that, on the left, the conjugate method converges in exactly two iterations, while coordinate descent takes several steps before finding the minimum. On the right, both methods become equivalent, since, when $H = I$, the coordinate directions become also conjugate to each other.

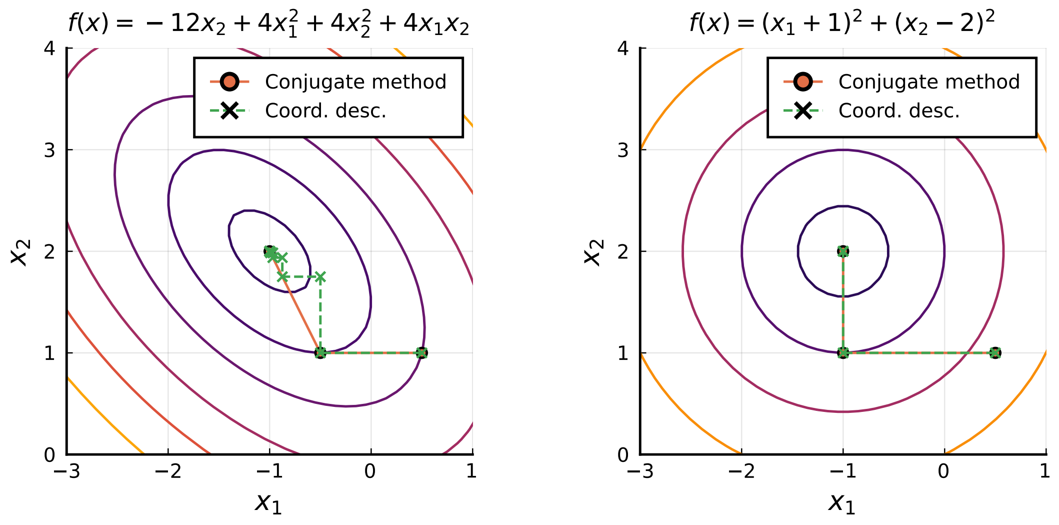

Figure 2: Optimising f with the conjugate method and coordinate descent (left). For $H = I$, both methods coincide (right)

1.1.2 Generating conjugate directions

The missing part at this point is how one can generate H-conjugate directions. This can be done efficiently using an adaptation of the *Gram-Schmidt* procedure, typically employed to generate orthonormal bases.

We intend to build a collection of conjugate directions $d_0, \dots, d_{n-1}$, which can be achieved provided that we have a collection of linearly independent vectors $\xi_0, \dots, \xi_{n-1}$.

The method proceed as follows.

1. First, set $d_0 = \xi_0$ as a starting step.
2. At a given iteration $k+1$, we need to set the coefficients α^i_{k+1} such that d_{k+1} is H-conjugate to $d_0, \dots, d_k$ and formed by adding ξ_{k+1} to a linear combination of $d_0, \dots, d_k$, that is

$$d_{k+1} = \xi_{k+1} + \sum_{l=0}^{k} \alpha^l_{k+1} d_l.$$

3. To obtain H-conjugacy one must observe that, for each $i = 0, \dots, k$,

$$d_{k+1}{}^\top H d_i = \xi_{k+1}{}^\top H d_i + \left(\sum_{l=0}^{k} \alpha^l_{k+1} d_l \right)^\top H d_i = 0.$$

Due to the H-conjugacy, $d_l^\top H d_k = 0$ for all $l \neq k$. Thus the value of α_{k+1} is

$$\alpha_{k+1}^i = \frac{-\xi_{k+1}^\top H d_i}{d_i^\top H d_i}, \text{ for } i = 0, \dots, k. \tag{1}$$

1.1.3 Gradients and conjugate directions

The next piece required for developing a method that could exploit conjugacy is the definition of what collection of linearly independent vectors $\xi_0, \dots, \xi_{n-1}$ could be used to generate conjugate directions. In the setting of developing an unconstrained optimisation method, the gradients $\nabla f(x_k)$ can play this part, which is the key result in Theorem 2.

Theorem 2. *Let $f(x) = c^\top x + \frac{1}{2}x^\top Hx$, where H is an $n \times n$ symmetric matrix. Let $d_1, \dots, d_n$ be H-conjugate, and let x_0 be an arbitrary starting point. Let λ_j be the optimal solution to $F_j(\lambda_j) = f(x_0 + \lambda_j d_j)$ for all $j = 1, \dots, n$. Then, for $k = 1, \dots, n$ we must have:*

1. *x_{k+1} is optimal to min. $\{f(x) : x - x_0 \in L(d_1, \dots, d_k)\}$ where $L(d_1, \dots, d_k) = \left\{\sum_{j=1}^k \mu_j d_j : \mu_j \in \mathbb{R}, j = 1, \dots, k\right\}$;*
2. *$\nabla f(x_{k+1})^\top d_j = 0$, for all $j = 1, \dots, k$;*

The proof of this theorem is based on the idea that, for a given collection of conjugate directions $d_0, \dots, d_k$, x_k will be optimal in the space spanned by the conjugate directions $d_0, \dots, d_k$, meaning that the partial derivatives of $F(\lambda)$ for these directions is zero. This phenomena is sometimes called *the expanding manifold property*, since at each iteration $L(d_0, \dots, d_k)$ expands in one independent (conjugate) direction at the time. To verify the second point, notice that the optimality condition for $\lambda_j \in \arg\min \{F_j(\lambda_j)\}$ is $d_j^\top \nabla f(x_0 + \lambda d_j) = 0$.

1.1.4 Conjugate gradient method

We have now all parts required for describing the *conjugate gradient method*. The method uses the gradients $\nabla f(x_k)$ as linearly independent vectors to generate conjugate directions, which are then used as search directions d_k.

In specific, the method operates generating a sequence of iterates

$$x_{k+1} = x_k + \lambda_k d_k,$$

where $d_0 = -\nabla f(x_0)$. Given a current iterate x_{k+1} with $-\nabla f(x_{k+1}) \neq 0$, we use Gram-Schmidth procedure, in particular (1), to generate a conjugate direction d_{k+1} by making the linearly independent vector $\xi_{k+1} = \nabla f(x_{k+1})$. Thus, we obtain

$$d_{k+1} = -\nabla f(x_{k+1}) + \alpha_k d_k, \text{ with } \alpha_k = \frac{\nabla f(x_{k+1})^\top H d_k}{d_k^\top H d_k}. \tag{2}$$

Notice that, since $\nabla f(x_{k+1}) - \nabla f(x_k) = H(x_{k+1} - x_k) = \lambda_k H d_k$ and $d_k = -\nabla f(x_k) + \alpha_{k-1} d_{k-1}$, α_k can be simplified to be

$$\begin{aligned}
\alpha_k &= \frac{\nabla f(x_{k+1})^\top H d_k}{d_k^\top H d_k} \\
&= \frac{\nabla f(x_{k+1})^\top (\nabla f(x_{k+1}) - \nabla f(x_k))}{(-\nabla f(x_k) + \alpha_{k-1} d_{k-1})^\top (\nabla f(x_{k+1}) - \nabla f(x_k))} \\
&= \frac{||\nabla f(x_{k+1})||^2}{||\nabla f(x_k)||^2},
\end{aligned}$$

where the last relation follows from Theorem 2. Algorithm 1 summarises the conjugate gradient method.

Algorithm 1 Conjugate gradient method

```
1: initialise. tolerance $\epsilon > 0$, initial point $x_0$, direction $d_0 = -\nabla f(x_0)$, $k = 1$
2: while $||\nabla f(x_k)|| > \epsilon$ do
3:     $y_0 = x_{k-1}$
4:     $d_0 = -\nabla f(y_0)$
5:     for $j = 1, \dots, n$ do
6:         $\overline{\lambda}_j = \text{argmin}_{\lambda \geq 0} \{f(y_{j-1} + \lambda d_{j-1})\}$
7:         $y_j = y_{j-1} + \overline{\lambda}_j d_{j-1}$
8:         $d_j = -\nabla f(y_j) + \alpha_j d_{j-1}$, where $\alpha_j = \frac{||\nabla f(y_j)||^2}{||\nabla f(y_{j-1})||^2}$.
9:     end for
10:    $x_k = y_n$, $k = k + 1$
11: end while
12: return $x_k$.
```

The conjugate gradient method using $\alpha_k = \frac{||\nabla f(x_{k+1})||^2}{||\nabla f(x_k)||^2}$ is due to Fletcher and Reeves. An alternative version of the method uses

$$\alpha_k = \frac{\nabla f(x_{k+1})^\top (\nabla f(x_{k+1}) - \nabla f(x_k))}{||\nabla f(x_k)||},$$

which is known for having better numerical properties for solving problems that are not quadratic.

Figure 3 illustrates the behaviour of the conjugate gradient method when applied to solve $f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$ using both exact and inexact line searches.

If $f : \mathbb{R}^n \mapsto \mathbb{R}$ is a quadratic function, then the method is guaranteed to converge in exactly n iterations. However, the method can be applied to any differentiable function f, in which setting the method behaves as successively solving quadratic approximations of f, in a similar fashion to that of Newton's method, but without requiring second-order (Hessian) information, which is the most demanding aspect associated with Newton's method. When employed to non-quadratic functions, the process of obtaining conjugate directions is restarted at the current point x_k after n steps (represented in the loop staring in Line 5 in Algorithm 1).

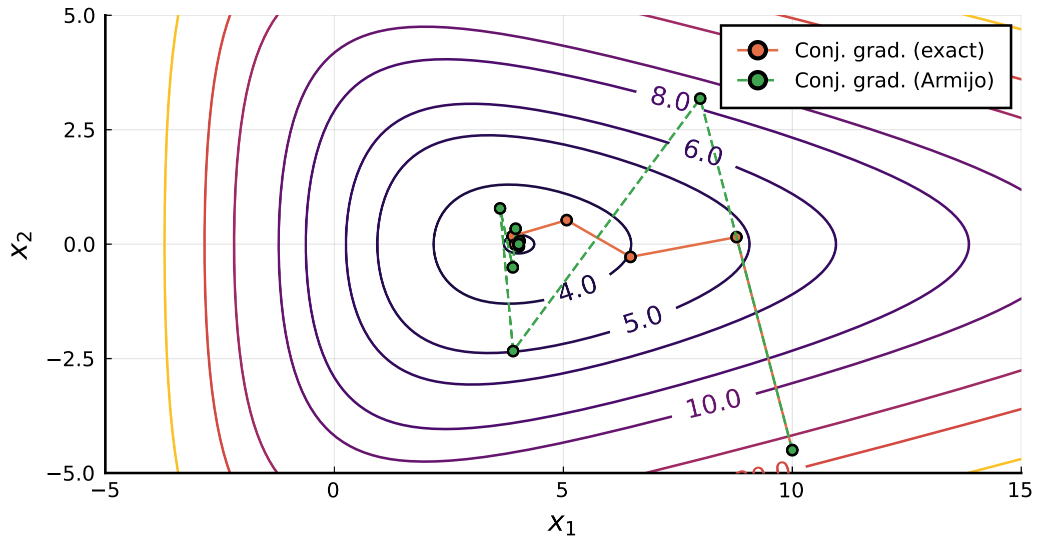

Figure 3: Conjugate gradient method applied to f. Convergence is observed in 24 steps using exact line search and 28 using Armijo's rule ($\epsilon = 10^{-6}$)

Equation (2) exposes an important property of the conjugate gradient method. In general, the employment of second-order terms is helpful for the optimisation method because it encodes *curvature information* on the definition of the search direction. The conjugate gradient method is also capable of encoding curvature information, not by using Hessians, but by weighting the current direction (given by the gradient) $-\nabla f(x_{k+1})$ and the previous direction $\alpha_k d_k$, which naturally compensates for the curvature encoded in the original matrix H (which is the Hessian of the quadratic approximation).

1.2 Quasi Newton: BFGS method

Quasi-Newton methods is a term referring to methods that use approximations for the inverse of the Hessian of f at $\overline{x}$, $H^{-1}(\overline{x})$, that do not explicitly require second-order information (i.e., Hessians) neither expensive inversion operations.

In quasi-Newton methods, we consider the search direction $d_k = -D_k \nabla f(x_k)$, where D_k acts as the approximation for the inverse Hessian $H^{-1}(\overline{x})$. To compute D_k, we use local curvature information, in the attempt to approximate second-order derivatives. For that, let us define the terms

$$\begin{aligned} p_k &= \lambda_k d_k = x_{k+1} - x_k \\ q_k &= \nabla f(x_{k+1}) - \nabla f(x_k) = H(x_{k+1} - x_k) = Hp_k. \end{aligned}$$

Starting from an initial guess D_0, quasi-Newton methods progress by successively updating $D_{k+1} = D_k + C_k$, with C_k being such that it only uses the information in p_k and q_k and that, after n updates,

D_n converges to H^{-1}.

For that to be the case, we require that $p_j,\ j = 1, \dots, k$ are eigenvectors of $D_{k+1}H$ with unit eigenvalue, that is

$$D_{k+1}Hp_j = p_j, \text{ for } j = 1, \dots, k. \tag{3}$$

This condition guarantees that, at the last iteration, $D_n = H^{-1}$. To see that, first, notice the following from (3).

$$\begin{aligned}
&D_{k+1}Hp_j = p_j,\ j = 1, \dots, k\\
&D_{k+1}q_j = p_j,\ j = 1, \dots, k\\
&D_kq_j + C_kq_j = p_j\ j = 1, \dots, k\\
&p_j = D_kHp_j + C_kq_j = p_j + C_kq_j,\ j = 1, \dots, k-1,
\end{aligned}$$

which implies that $C_kq_j = 0$ for $j = 1, \dots, k-1$.

Now, for $j = k$, we require that

$$\begin{aligned}
&D_{k+1}q_k = p_k\\
&D_kq_k + C_kq_k = p_k\\
&(D_k + C_k)q_k = p_k
\end{aligned}$$

This last condition allows, after n iterations, to recover

$$D_n = [p_0, \dots, p_{n-1}][q_0, \dots, q_{n-1}]^{-1} = H(x_n) \tag{4}$$

Condition (4) is called the *secant condition* as a reference to the approximation to the second-order derivative. Another way of understanding the role this condition has is by noticing the following.

$$\begin{aligned}
D_{k+1}q_k &= p_k\\
D_{k+1}(\nabla f(x_{k+1}) - \nabla f(x_k)) &= x_{k+1} - x_k\\
\nabla f(x_{k+1}) &= \nabla f(x_k) + D_{k+1}^{-1}(x_{k+1} - x_k),
\end{aligned} \tag{5}$$

where D_{k+1}^{-1} can be seen as an approximation to the Hessian H, just as D_{k+1} is an approximation to H^{-1}. Now, consider the second-order approximation of f at x_k

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

We can now notice the resemblance the condition (5) holds with

$$\nabla q(x) = \nabla f(x_k) + H(x_k)^\top (x - x_k) = 0.$$

In other words, at each iteration, the updates are made such that the optimality conditions in terms of the quadratic expansion remains valid.

The *Davidon-Fletcher-Powell* (DFP) is one classical quasi-Newton method available. It employs updates of the form

$$D_{k+1} = D_k + C^{DFP} = D_k + \frac{p_k p_k^\top}{p_k^\top q_k} - \frac{D_k q_k q_k^\top D_k}{q_k^\top D_k q_k}$$

We can verify that C^{DFP} satisfies conditions (3) and (4). For that, notice that

(1) $C^{DFP} q_j = C^{DFP} H p_j$

$= \frac{p_k p_k^\top H p_j}{p_k^\top q_k} - \frac{D_k q_k p_k^\top H D_k H p_j}{q_k^\top D_k q_k} = 0,$ for $j = 1, \dots, k-1$;

(2) $C^{DFP} q_k = \frac{p_k p_k^\top q_k}{p_k^\top q_k} - \frac{D_k q_k q_k^\top D_k q_k}{q_k^\top D_k q_k} = p_k - D_k q_k.$

The main difference between available quasi-Newton methods is the nature of the matrix C employed in the updates. Over the years, several ideas emerged in terms of generating updates that satisfied the above properties. The most widely used quasi-Newton method is the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS), which has been widely shown to have remarkable practical performance. BFGS is part of the Broyden family of updates, given by

$$C^B = C^{DFP} + \phi \frac{\tau_j v_k v_k^\top}{p_k^\top q_k},$$

where $v_k = p_k - \left(\frac{1}{\tau_k}\right) D_k q_k$, $\tau_k = \frac{q_j^\top D_k q_k}{p_k^\top q_k}$, and $\phi \in (0, 1)$. The extra term in the Broyden family of updates is designed to help with mitigating numerical difficulties from near-singular approximations.

It can be shown that all updates from the Broyden family also satisfy the quasi-Newton conditions (3) and (4). The BFGS update is obtained for $\phi = 1$, which renders

$$C_k^{BFGS} = \frac{p_k p_k^\top}{p_k^\top q_k} \left(1 + \frac{q_k^\top D_k q_k}{p_k^\top q_k}\right) - \frac{D_k q_k p_k^\top + p_k q_k^\top D_k}{p_k^\top q_k}.$$

The BFGS method is often presented explicitly approximating the Hessian H instead of its inverse, which is useful when using specialised linear algebra packages that rely on the "backslash" operator to solve linear systems of equations. Let B_k be the current approximation of H. Then $D_{k+1} =$

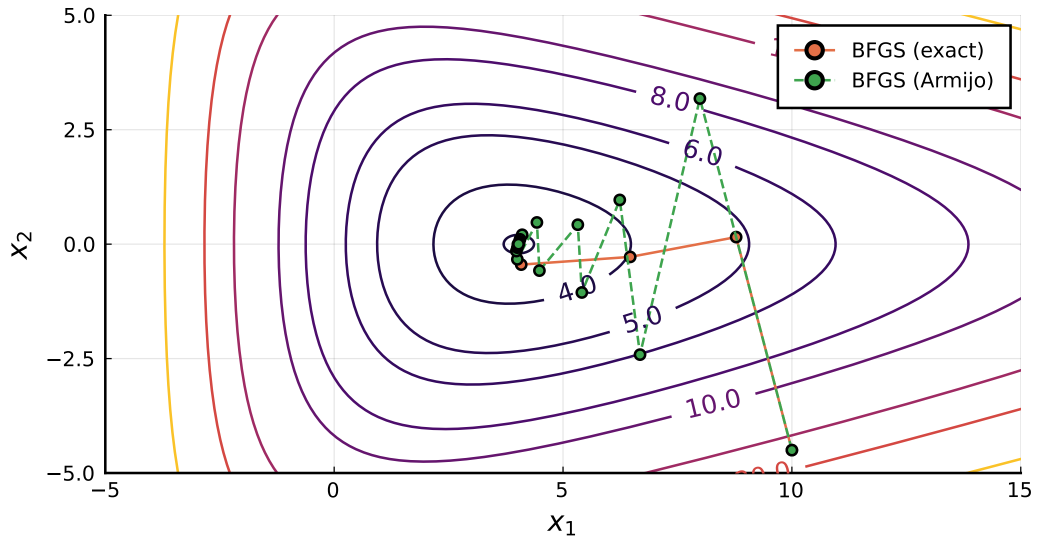

Figure 4: BFGS method applied to f. Convergence is observed in 11 steps using exact line search and 36 using Armijo's rule ($\epsilon = 10^{-6}$)

$B_{k+1}^{-1} = (B_k + \overline{C}_k^{BFGS})^{-1}$, with

$$\overline{C}_k^{BFGS} = \frac{q_k q_k^\top}{q_k^\top p_k} - \frac{B_k p_k p_k^\top B_k}{p_k^\top B_k p_k}.$$

The update for the inverse Hessian H^{-1} can then be obtained using the *Sherman-Morrison formula*.

Figure 4 illustrates the behaviour of the BFGS method when applied to solve

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

using both exact and inexact line searches. Notice how the combination of imprecisions both in the calculation of H^{-1} and in the line search turns the search noisy. This combination (BFGS combined with Armijo rule) is, however, widely used in efficient implementations of several nonlinear optimisation methods.

A variant of BFGS, called the *limited memory* BFGS (l-BFGS) utilises efficient implementations that do not require storing the whole approximation for the Hessian, but only a few most recent p_k and q_k vectors.

2 Complexity, convergence and conditioning

Several aspects must be considered when analysing the performance of algorithms under a given setting and, in each, a multitude of theoretical results that can be used to understand, even if to some extent, the performance of a given optimisation method.

We focus on three key properties that one should be aware when employing the methods we have seen to solve optimisation problems. The first two, *complexity* and *convergence* refer to the algorithm itself, but often involve considerations related to the function being optimised. *Conditioning*, on the other hand, is a characteristic exclusively related to the problem at hand. Knowing how the "three C's" can influence the performance of an optimisation problem is central in making good choices in terms of which optimisation method to employ.

2.1 Complexity

Algorithm complexity analysis is a discipline from computer science that focus on deriving worst-case guarantees in terms of the number of computational steps required for an algorithm to converge, given an input of known size. For that, we use the following definition to identify efficient, generally referred to as *polynomial*, algorithms.

Definition 3 (Polynomial algorithms)**.** *Given a problem P, a problem instance $X \in P$ with length $L(X)$ in binary representation, and an algorithm A that solves X, let $f_A(X)$ be the number of* elementary calculations *required to run A on X. Then, the running time of A on X is proportional to*

$$f_A^*(n) = \sup_X \{f_A(X) : L(X) = n\} .$$

Algorithm A is polynomial *for a problem P if $f_A^*(n) = O(n^p)$ for some integer p.*

Notice that this sort of analysis only render bounds on the worst-case performance. Though it can be informative under a general setting, there are several well known examples in that experimental practice does not correlate with the complexity analysis. One famous example is the simplex method for linear optimisation problems, which despite not being a polynomial algorithm, presents widely-demonstrated reliable (polynomial-like) performance.

2.2 Convergence

In the context of optimisation, *local analysis* is typically more informative regarding to the behaviour of optimisation methods. This analysis tend to disregard initial steps further from the initial points and concentrate on the behaviour of the sequence $\{x_k\}$ to a unique point $\overline{x}$.

The convergence is analysed by means of *rates of convergence* associated with *error functions* $e : \mathbb{R}^n \mapsto \mathbb{R}$ such that $e(x) \geq 0$. Typical choices for e include:

- $e(x) = ||x - \overline{x}||$;
- $e(x) = |f(x) - f(\overline{x})|$.

The sequence $\{e(x_k)\}$ is then compared to the geometric progression β^k, with $k = 1, 2, \dots,$ and $\beta \in (0, 1)$. We say that a method presents *linear convergence* if exists $q > 0$ and $\beta \in (0, 1)$ such that $e(x) \leq q\beta^k$ for all k. An alternative way of posing this result is stating that

$$\lim_{k \to \infty} \sup \frac{e(x_{k+1})}{e(x_k)} \leq \beta.$$

We say that an optimisation method converges superlinearly if the rate of convergence tends to zero. That is, if exists $\beta \in (0, 1)$, $q > 0$ and $p > 1$ such that $e(x_k) \leq q\beta^{p^k}$ for all k. For $k = 2$, we say that the method presents quadratic convergence. Any p-order convergence is obtained if

$$\lim_{k \to \infty} \sup \frac{e(x_{k+1})}{e(x_k)^p} < \infty, \text{ which is true if } \lim_{k \to \infty} \sup \frac{e(x_{k+1})}{e(x_k)} = 0.$$

Linear convergence is the most typical convergence rate for nonlinear optimisation methods, which is satisfactory if β is not too close to one. Certain methods are capable of achieving superlinear convergence for certain problems, being Newton's method an important example.

In light of what we discussed, let us analyse the convergence rate of some of the methods earlier discussed. We start by posing the convergence of gradient methods.

Theorem 4 (Convergence of the gradient method)**.** *Let $f(x) = \frac{1}{2}x^\top Hx$ where H is a positive definite symmetric matrix. Suppose $f(x)$ is minimised with the gradient method using an exact line search. Let $\underline{\lambda} = \min_{i=1,\dots,n} \lambda_i$ and $\overline{\lambda} = \max_{i=1,\dots,n} \lambda_i$, where λ_i are eigenvalues of H. Then, for all k,*

$$\frac{f(x_{k+1})}{f(x_k)} \leq \left(\frac{\overline{\lambda} - \underline{\lambda}}{\overline{\lambda} + \underline{\lambda}}\right)^2$$

Theorem 4 implies that, under certain assumptions, the gradient methods present *linear convergence*. Moreover, this result shows that the convergence rate is *dependent* on the scaling of the function, since it depends on the ratio of eigenvalues of H, which in turn can be modified by scaling f. This results exposes an important shortcoming that gradient methods present: the dependence on the *conditioning* of the problem, which we will discuss shortly. Moreover, this result can be extended to incorporate functions other than quadratic and also inexact line searches.

The convergence of Newton's method is also of interest since, under specific circumstances, it presents a quadratic convergence rate. Theorem 5 summarises these conditions.

Theorem 5 (Convergence of Newton's method - general case)**.** *Let $g : \mathbb{R}^n \to \mathbb{R}^n$ be differentiable, $\overline{x}$ such that $g(\overline{x}) = 0$, and let $\{e(x_k)\} = \{||x_k - \overline{x}||\}$. Moreover, let $N_\delta(\overline{x}) = \{x : ||x - \overline{x}|| \leq \delta\}$ for some $\delta > 0$. Then*

1. *There exists $\delta > 0$ such that if $x_0 \in N_\delta(\overline{x})$, the sequence $\{x_k\}$ with $x_{k+1} = x_k - (\nabla g(x_k)^\top)^{-1} g(x_k)$ belongs to $N_\delta(\overline{x})$ and converges to $\overline{x}$, while $\{e(x_k)\}$ converges superlinearly.*
2. *If for some $L > 0$, $M > 0$, and for all $x, y \in N_\delta(\overline{x})$, $\lambda \in (0, \delta]$*

$$\nabla g(x) - \nabla g(y) \leq L||x - y|| \quad \text{and} \quad ||(\nabla g(x_k)^\top)^{-1}|| \leq M,$$

then, if $x_0 \in N_\delta(\overline{x})$, we have for $k = 0, 1, \dots$

$$||x_{k+1} - \overline{x}|| \leq \frac{LM}{2}||x_k - \overline{x}||^2.$$

If $\frac{LM\delta}{2} < 1$ and $x_0 \in N_\delta(\overline{x})$, $\{e(x_k)\}$ converges quadratically.

Notice that the convergence of the method is analysed in two distinct phases. In the first phase, referred to as 'damped' phase, superlinear convergence is observed within the neighbourhood $N_\delta(\overline{x})$ defined by δ. The second phase is where quadratic convergence is observed and it happens when $\delta < \frac{2}{LM}$, which in practice can only be interpreted as small enough, as the constants L (the Lipschitz constant) and M (a finite bound for the norm of the Hessian) cannot be easily estimated in practical applications.

However, it is interesting to notice that the convergence result for Newton's method do not depend on the scaling of the problem, like the gradient method. This property, called *affine invariance* is one of the greatest features that Newton's method possess.

Figure 5 compare the convergence of four methods presented considering $f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$, employing exact line search and using $e(x) = ||x_k - \overline{x}||$. Notice how the quadratic convergence of Newton's method compare with the linear convergence of the gradients method. The other two, conjugate gradients and BFGS, present superlinear convergence.

2.3 Conditioning

The *condition number* of a symmetric matrix is given by

$$\kappa = ||A||_2||A^{-1}||_2 = \frac{\max_{i=1,\dots,n}\{\lambda_i\}}{\min_{i=1,\dots,n}\{\lambda_i\}} = \frac{\overline{\lambda}}{\underline{\lambda}}$$

The condition number κ is an important measure in optimisation, since it can be used to predict how badly scaled a problem might be. Large κ values mean that numerical errors will be amplified after repeated iterations, in particular matrix inversions.

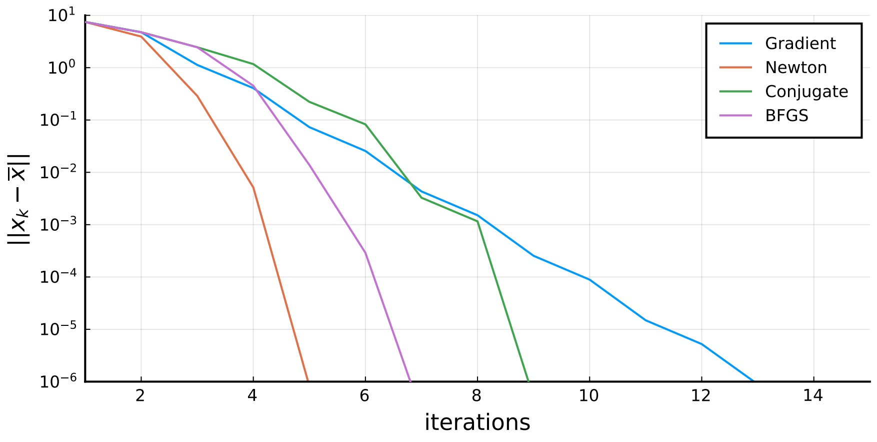

Figure 5: Convergence comparison for the four methods

Roughly speaking, having $\kappa \geq 10^k$ means that at each iteration, k digits of accuracy are lost. As general rule, one would prefer smaller κ numbers, but good values are entirely problem dependent.

One way of understanding the role that the conditioning number κ has is to think the role that the eigenvalues of the Hessian have in the shape of the level curves of quadratic approximations of a general function $f : \mathbb{R}^n \mapsto \mathbb{R}$. First, let us consider the Hessian $H(x)$ at a given point $x \in \mathbb{R}^n$ is the identity matrix I, for which all eigenvalues are 1 and eigenvectors are e_i, $i = 1, \dots, n$, where e_i is the vector with component 1 in the position i and zero everywhere else. This means that in the direction of the n-eigenvectors, the ellipsoid formed by the level curves (specifically, the lower level sets) of f stretch by the same magnitude and, therefore, the level curves of the quadratic approximation are in fact a circle. Now, suppose that for one of the dimensions i of the matrix $H(x)$, we have one of the eigenvalues greater than 1. What we would see is that the level curves of the quadratic approximation will be more stretched in that dimension i than in the others. The reason for that is because the Hessian plays a role akin to that of a characteristic matrix in an ellipsoid (specifically due to the second order term $\frac{1}{2}(x - x_k)^\top H(x_k)(x - x_k)$ in the quadratic approximation).

Thus, larger κ will mean that the ratio between the eigenvalues is larger, which in turn implies that there is eccentricity in the lower level sets (i.e., the lower level sets are far wider in one direction than in others), which ultimately implies that first-order methods struggle since often the gradients often point to directions that only show descent for small step sizes.

Figure 6 illustrates the effect of different condition numbers on the performance of the gradient method. As can be seen, the method require more iterations for higher conditioning numbers, in accordance to the convergence result presented in Theorem 4.

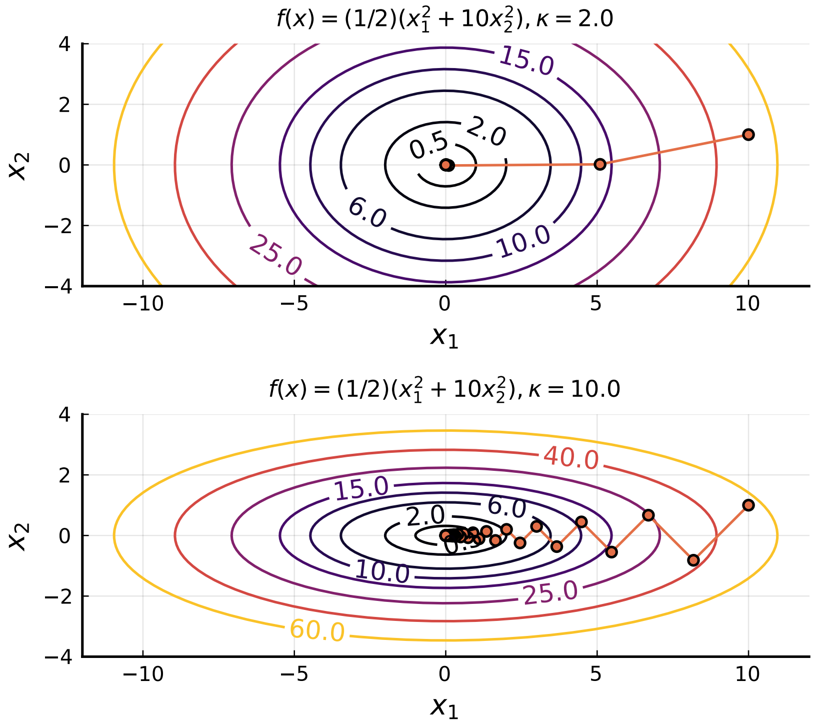

Figure 6: The gradient method with exact line search for different κ.