
MSE2122 - Nonlinear Optimization Extra Material

Fernando Dias (based on previous version by
Fabricio Oliveira)

October 26, 2023

Abstract

Main takeaways from Lecture I to VI.

Contents

| | | |
|----------|--|----------|
| 1 | Most important concepts | 2 |
| 1.1 | Introduction | 2 |
| 1.2 | Convexity | 2 |
| 1.3 | Weierstrass theorem | 3 |
| 1.4 | Convexity in functions | 3 |
| 1.5 | Differentiability | 5 |
| 1.6 | Optimality Conditions | 5 |
| 1.6.1 | First-order optimality conditions | 5 |
| 1.6.2 | Second-order optimality conditions | 6 |
| 2 | Most important models | 8 |
| 2.1 | Basic algorithms | 8 |
| 2.2 | Linear search methods | 8 |
| 2.3 | Descent methods | 8 |

1 Most important concepts

1.1 Introduction

In this course, the focus is on **optimisation**.

In **mathematical optimisation**, we build upon concepts and techniques from basic mathematical subfields. From **calculus**, **analysis**, **linear algebra**, and other, we to model and develop **methods** that allow us to find values for variables within a given domain that **maximise** (or **minimise**) the value of a **function**. In specific, we are trying to solve the following general problem:

$$\begin{aligned} \min. & f(x) \\ \text{subject to: } & x \in X. \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^n$ is a vector of n variables, $f : \mathbb{R}^n \mapsto \mathbb{R}$ is a **function** to be optimised (minimised) and $X \subseteq \mathbb{R}^n$ is a **domain** containing **potential** values for x .

Let us separate **mathematical programming** from (mathematical) **optimization**. Mathematical programming is a modelling paradigm in which we rely on (compelling) analogies to model *real-world* problems. In that, we look at a given decision problem considering that:

- **variables** represent *decisions* or *interest*, as in a business decision or a course of action. Examples include setting the parameter of (e.g., prediction) model, production systems layouts, geometries of structures, topologies of networks, and so forth;
- **domain** represents business rules or *constraints* and *limitations*, representing logic relations, design or engineering limitations, requirements, and such;
- function is an *objective function* that measures solution quality, profit or goal.

In general, **the rule of thumb** for solving a problem is :

The simpler the assumptions on the parts forming the optimisation model, the more efficient the methods for solving it.

Let us define some additional notation that we will use from now on. Consider a model in the **general (or standard) form**:

$$\begin{aligned} \min. & f(x) \\ \text{subject to: } & g_i(x) \leq 0, i = 1, \dots, m \\ & h_i(x) = 0, i = 1, \dots, l \\ & x \in X, \end{aligned}$$

More details in **Lecture I**

1.2 Convexity

In order to understand and apply different optimisation techniques, we first need to understand **convexity**.

In a nutshell, convexity allows us to infer the **global properties** of a solution (i.e., that holds for all of its domain) by considering exclusively local information (such as gradients, for example). Such property is critical in **optimization** since most methods we know to perform well in practice are designed to find solutions that satisfy local optimality conditions. Once convexity is attested, one can then guarantee that these local solutions are, in fact, **globally optimal** without exhaustively exploring the solution space.

For a problem of the form:

$$\begin{aligned} (P) : \min. & f(x) \\ \text{subject to: } & x \in X \end{aligned}$$

to be convex, we need to verify whether f is a **convex function** and X is a **convex set**. If both statements hold, we can conclude that P is a **convex problem**. We start looking into how to identify

convex sets since we can use the convexity of sets to infer the convexity of functions.

We say a set is convex if it contains all points formed by the convex combination of any pair of points in this set, which is equivalent to saying that the set contains the line segment between any two points belonging to the set.

Definition 1.1. Convex sets

A set $S \subseteq \mathbb{R}^n$ is said to be convex if $\bar{x} = \sum_{j=1}^k \lambda_j x_j$ belongs to S , where $\sum_{j=1}^k \lambda_j = 1$, $\lambda_j \geq 0$ and $x_j \in S$ for $j = 1, \dots, k$.

Complete explanation and details in **Lecture II**

1.3 Weierstrass theorem

The **Weierstrass theorem** is a result that guarantees the existence of optimal solutions for optimization problems. To make it more precise, let:

$$(P) : z = \min. \{f(x) : x \in S\}$$

be our optimization problem. If an optimal solution x^* exists, then $f(x^*) \leq f(x)$ for all $x \in S$ and $z = f(x^*) = \min\{f(x) : x \in S\}$.

Notice the difference between $\min.$ (an abbreviation for **minimize**) and the operator \min . The first is meant to represent the problem of **minimizing** the function f in the domain S , while \min is shorthand for **minimum**, in this case z , assuming it is attainable.

It might be that an optimal solution is not attainable, but a bound can be obtained for the optimal solution value. The greatest lower bound for z is its **infimum** (or **supremum** for maximization problems), denoted by \inf . That is, if $z = \inf\{f(x) : x \in S\}$, then $z \leq f(x)$ for all $x \in S$ and there is no $\bar{z} > z$ such that $\bar{z} \leq f(x)$ for all $x \in S$. We might sometimes use the notation:

$$(P) : z = \inf\{f(x) : x \in S\}$$

to represent optimization problems for which one cannot be sure whether an optimal solution is attainable. The Weierstrass theorem describes the situations in which those minimums (or maximums) are guaranteed to be attained, which is the case whenever S is compact.

Theorem 1.2. Weierstrass theorem

Let $S \neq \emptyset$ be a compact set, and let $f : S \rightarrow \mathbb{R}$ be continuous on S . Then there exists a solution $\bar{x} \in S$ to $\min. \{f(x) : x \in S\}$.

In completion in **Lecture II**.

1.4 Convexity in functions

Now, we turn our attention to identifying the convexity of functions. Consider the general problem:

$$\begin{aligned} (P) : & \min. f(x) \\ & \text{subject to: } g(x) \leq 0 \\ & x \in X \end{aligned}$$

with $f : \mathbb{R}^n \mapsto \mathbb{R}$, $g : \mathbb{R}^n \mapsto \mathbb{R}^m$ and $X \subseteq \mathbb{R}^n$. Assuming X is a convex set, the next step towards attesting that (P) is a convex problem is to check whether f and g are convex. It is important to textbfasise (perhaps redundantly at this point) how crucial it is for us to attest to the convexity (P) since it allows us to generalise local optimality results to the whole domain of the problem.

The convexity of functions has a different definition than that used to define convex sets.

Definition 1.3. Convexity of a function I

Let $f : S \mapsto \mathbb{R}$ where $S \subseteq \mathbb{R}^n$ is a nonempty convex set. The function f is said to be **convex** on S if

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$$

for each $x_1, x_2 \in S$ and for each $\lambda \in [0, 1]$.

Convexity can be also expressed via lower level set and epigraph. Let us first consider **level sets**, one type of set spawned by functions.

Definition 1.4. Lower level set

Let $S \subseteq \mathbb{R}^n$ be a nonempty set. The lower level set of $f : \mathbb{R}^n \mapsto \mathbb{R}$ for $\alpha \in \mathbb{R}$ is given by

$$S_\alpha = \{x \in S : f(x) \leq \alpha\}.$$

Figure 1 illustrates the lower-level sets of two functions. The lower level set S_α can be seen as the projection of the function image onto the domain for a given level α .

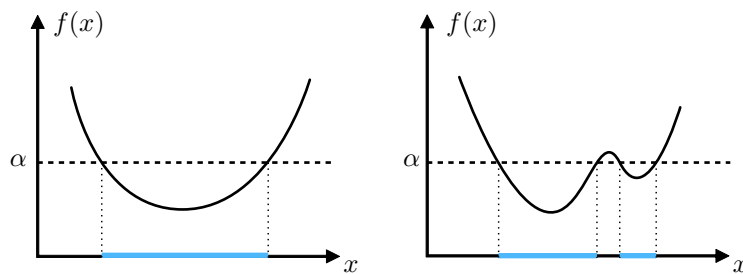


Figure 1: The lower level sets S_α (in blue) of two functions, given a value of α . Notice the nonconvexity of the level set of the nonconvex function (on the right)

Epigraphs, on the other hand, can be used to show the convexity of functions. Let us first formally define epigraphs.

Definition 1.5. Epigraph

Let $S \subseteq \mathbb{R}^n$ be a nonempty set and $f : S \mapsto \mathbb{R}$. The epigraph of f is

$$\mathbf{epi}(f) = \{(x, y) : x \in S, y \in \mathbb{R}, y \geq f(x)\} \subseteq \mathbb{R}^{n+1}$$

Figure 2 illustrates the epigraphs of two functions. Notice that the second function (on the right) is neither convex nor epigraph. We can use the convexity of epigraphs (and the technical results associated with the convexity of sets) to show the convexity of functions.

Theorem 1.6. Convex epigraphs

Let $S \subseteq \mathbb{R}^n$ be a nonempty convex set and $f : S \mapsto \mathbb{R}$. Then f is convex if and only if $\mathbf{epi}(f)$ is a convex set.

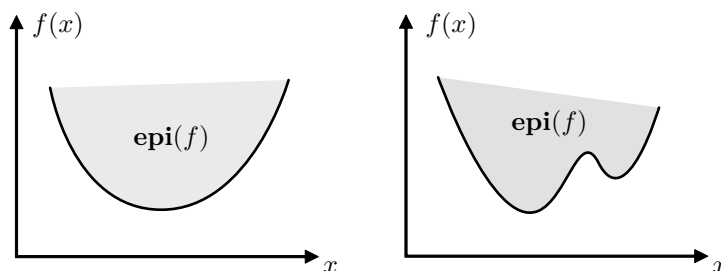


Figure 2: The epigraph $\mathbf{epi}(f)$ of a convex function is a convex set (in grey on the left).

Proof. First, suppose f is convex and let $(x_1, y_1), (x_2, y_2) \in \mathbf{epi}(f)$ for $\lambda \in (0, 1)$. Then

$$\lambda y_1 + (1 - \lambda)y_2 \geq \lambda f(x_1) + (1 - \lambda)f(x_2) \geq f(\lambda x_1 + (1 - \lambda)x_2).$$

As $\lambda x_1 + (1 - \lambda)x_2 \in S$, $(\lambda x_1 + (1 - \lambda)x_2, \lambda y_1 + (1 - \lambda)y_2) \in \mathbf{epi}(f)$.

Conversely, suppose $\mathbf{epi}(f)$ is convex. Therefore $x_1, x_2 \in S$: $(x_1, f(x_1)) \in \mathbf{epi}(f)$, $(x_2, f(x_2)) \in \mathbf{epi}(f)$ and $(\lambda x_1 + (1 - \lambda)x_2, \lambda f(x_1) + (1 - \lambda)f(x_2)) \in \mathbf{epi}(f)$ for $\lambda \in (0, 1)$, implying that $\lambda f(x_1) + (1 - \lambda)f(x_2) \geq f(\lambda x_1 + (1 - \lambda)x_2)$. 🤔

The proof starts with the implication “if f is convex, then $\mathbf{epi}(f)$ is convex”. For that, it assumes that f is convex and uses the convexity of f to show that any convex combination of x_1, x_2 in S will also be in the $\mathbf{epi}(f)$, which is the definition of a convex set.

To prove the implication “if $\mathbf{epi}(f)$ is convex, then f is convex”, we define a convex combination of points in $\mathbf{epi}(f)$ and use the definition of $\mathbf{epi}(f)$ to show that f is convex by setting $y = \lambda f(x_1) + (1 - \lambda)f(x_2)$ and $x = \lambda x_1 + (1 - \lambda)x_2$.

More details in **Lecture III**.

1.5 Differentiability

Besides convexity, differentiability is another important factor in optimisation.

Definition 1.7

Let $S \subseteq \mathbb{R}^n$ be a nonempty set. The function $f : S \mapsto \mathbb{R}$ is differentiable at $\bar{x} \in \mathbf{int}(S)$ if there exists a vector $\nabla f(\bar{x})$, called a gradient vector, and a function $\alpha : \mathbb{R}^n \mapsto \mathbb{R}$ such that

$$f(x) = f(\bar{x}) + \nabla f(\bar{x})^\top (x - \bar{x}) + \|x - \bar{x}\| \alpha(\bar{x}; x - \bar{x})$$

where $\lim_{x \rightarrow \bar{x}} \alpha(\bar{x}; x - \bar{x}) = 0$. If this is the case for all $\bar{x} \in \mathbf{int}(S)$, we say the function is differentiable in S .

The gradient is the first-order approximation while the Hessian is the second-order, if the function is twice differentiable.

We say that a function is **twice-differentiable** if it has a second-order Taylor expansion. Having second-order expansions can be useful. It allows for encoding curvature information in the approximation, which is characterised by the **Hessian**, and to verify convexity (or strict convexity) by testing for semi-definiteness (positive definiteness).

Let $f_{ij}(\bar{x}) = \frac{\partial^2 f(\bar{x})}{\partial x_i \partial x_j}$. Recall that the Hessian matrix $H(\bar{x})$ at \bar{x} is given by

$$H(\bar{x}) = \begin{bmatrix} f_{11}(\bar{x}) & \dots & f_{1n}(\bar{x}) \\ \vdots & \ddots & \vdots \\ f_{n1}(\bar{x}) & \dots & f_{nn}(\bar{x}) \end{bmatrix}$$

Second-order differentiability can be defined as follows.

Definition 1.8. Second-order differentiability

Let $S \subseteq \mathbb{R}^n$ be a nonempty set, and let $f : S \mapsto \mathbb{R}$. Then f is twice differentiable at $\bar{x} \in \mathbf{int}(S)$ if there exists a vector $\nabla f(\bar{x}) \in \mathbb{R}^n$, an $n \times n$ symmetric matrix $H(\bar{x})$ (the Hessian), and a function $\alpha : \mathbb{R}^n \mapsto \mathbb{R}$ such that

$$f(x) = f(\bar{x}) + \nabla f(\bar{x})^\top (x - \bar{x}) + \frac{1}{2} (x - \bar{x})^\top H(\bar{x}) (x - \bar{x}) + \|x - \bar{x}\|^2 \alpha(\bar{x}; x - \bar{x})$$

where $\lim_{x \rightarrow \bar{x}} \alpha(\bar{x}; x - \bar{x}) = 0$. If this is the case for all $\bar{x} \in S$, we say that the function is twice differentiable in S .

More details in **Lecture IV**.

1.6 Optimality Conditions

1.6.1 First-order optimality conditions


Let us start defining what it means to be a *descent direction*.

Theorem 1.9. Descent direction

Suppose $f : \mathbb{R}^n \mapsto \mathbb{R}$ is differentiable at \bar{x} . If there is d such that $\nabla f(\bar{x})^\top d < 0$, there exists $\delta > 0$ such that $f(\bar{x} + \lambda d) < f(\bar{x})$ for each $\lambda \in (0, \delta)$, so that d is a descent direction of f at \bar{x} .

Proof. By differentiability of f at \bar{x} , we have that:


$$\frac{f(\bar{x} + \lambda d) - f(\bar{x})}{\lambda} = \nabla f(\bar{x})^\top d + \|d\| \alpha(\bar{x}; \lambda d).$$

Since $\nabla f(\bar{x})^\top d < 0$ and $\alpha(\bar{x}; \lambda d) \rightarrow 0$ when $\lambda \rightarrow 0$ for some $\lambda \in (0, \delta)$, we must have $f(\bar{x} + \lambda d) - f(\bar{x}) < 0$. 

The proof uses the first-order expansion around \bar{x} to show that, f being differentiable, the condition $\nabla f(\bar{x})^\top d < 0$ implies that $f(\bar{x} + \lambda d) < f(\bar{x})$, or put in words, that a step in the direction d decreases the objective function value.

We can derive the first-order optimality condition as a consequence from Theorem 1.9. Notice, however, that since convexity is not assumed, all we can say is that this condition is necessary (but not sufficient) for local optimality.

Corollary 1.10 (First-order necessary condition). Suppose $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is differentiable at \bar{x} . If \bar{x} is a local minimum, then $\nabla f(\bar{x}) = 0$.

Proof. By contradiction, suppose that $\nabla f(\bar{x}) \neq 0$. Letting $d = -\nabla f(\bar{x})$, we have that $\nabla f(\bar{x})^\top d = -\|\nabla f(\bar{x})\|^2 < 0$. By Theorem 1.9, there exists a $\delta > 0$ such that $f(\bar{x} + \lambda d) < f(\bar{x})$ for all $\lambda \in (0, \delta)$, thus contradicting the local optimality of \bar{x} . 

Notice that Corollary 1.10 only holds in one direction. The proof uses contradiction once again, where we assume local optimality of \bar{x} and show that having $\nabla f(\bar{x}) \neq 0$ contradicts the local optimality of \bar{x} , our initial assumption. To do that, we simply show that having any descent direction d (we use $-\nabla f(\bar{x})$ since in this setting it is guaranteed to exist as $\nabla f(\bar{x}) \neq 0$) would mean that small step λ can reduce the objective function value, contradicting the local optimality of \bar{x} .

1.6.2 Second-order optimality conditions

We now derive necessary conditions for local optimality of \bar{x} based on second-order differentiability. As we will see, it requires that the Hessian $H(\bar{x})$ of $f(x)$ at \bar{x} is positive semidefinite.

Theorem 1.11. Second-order necessary condition

Suppose $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is twice differentiable at \bar{x} . If \bar{x} is a local minimum, then $H(\bar{x})$ is positive semidefinite.


Proof. Take an arbitrary direction d . As f is twice differentiable, we have:

$$f(\bar{x} + \lambda d) = f(\bar{x}) + \lambda \nabla f(\bar{x})^\top d + \frac{1}{2} \lambda^2 d^\top H(\bar{x}) d + \lambda^2 \|d\|^2 \alpha(\bar{x}; \lambda d)$$

since \bar{x} is a local minimum, Corollary 1.10 implies that $\nabla f(\bar{x}) = 0$ and $f(\bar{x} + \lambda d) \geq f(\bar{x})$.

Rearranging terms and dividing by $\lambda^2 > 0$ we obtain

$$\frac{f(\bar{x} + \lambda d) - f(\bar{x})}{\lambda^2} = \frac{1}{2} d^\top H(\bar{x}) d + \|d\|^2 \alpha(\bar{x}; \lambda d).$$

Since $\alpha(\bar{x}; \lambda d) \rightarrow 0$ as $\lambda \rightarrow 0$, we have that $d^\top H(\bar{x}) d \geq 0$. 

The second-order conditions can be used to attest local optimality of \bar{x} . In the case where $H(\bar{x})$ is positive definite, then this second order condition becomes *sufficient* for local optimality, since it implies that the function is 'locally convex' for a small enough neighbourhood $N_\epsilon(\bar{x})$.


In case f is convex, then the first-order condition $\nabla f(x) = 0$ becomes also sufficient for attesting the global optimality of \bar{x} . Recall that f is convex if and only if $H(x)$ is positive semidefinite for all $x \in \mathbb{R}^n$, meaning that in this case the second-order necessary conditions are also satisfied at \bar{x} .

Theorem 1.12

Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be convex. Then \bar{x} is a global minimum if and only if $\nabla f(\bar{x}) = 0$.

Proof. From Corollary 1.10, if \bar{x} is a global minimum, then $\nabla f(\bar{x}) = 0$. Now, since f is convex, we have that:

$$f(x) \geq f(\bar{x}) + \nabla f(\bar{x})^\top (x - \bar{x})$$

Notice that $\nabla f(\bar{x}) = 0$ implies that $\nabla f(\bar{x})^\top (x - \bar{x}) = 0$ for each $x \in \mathbb{R}^n$, thus implying that $f(\bar{x}) \leq f(x)$ for all $x \in \mathbb{R}^n$. 

Full details in **Lecture V**

2 Most important models

2.1 Basic algorithms

The rudimentary conceptual optimisation algorithm (as seen in **Lecture V**):

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 λ_k
 - 5: $x_{k+1} = x_k + \lambda_k d_k$
 - 6: $k = k + 1$
 - 7: **end while**
 - 8: **return** x_k .
-

In algorithm 1, it has two main elements, namely the computation of the **direction** d_k and the **step size** λ_k at each iteration k .

2.2 Linear search methods

For the step size, the strategy is divided into two main categories: exact and inexact search. For the former, it calculates the optimal step size by iteratively closing the gaps between two possible candidates in a pre-defined interval. The latter uses arbitrarily good approximations.

For exact, many algorithms have been proposed, such as **uniform** search, **dichotomous** search and **bisection** section. A special case is the **golden** search which uses the golden ratio as inverse rate of reduction for the search interval in each iteration.

For inexact, the most common strategy is to approximate the behaviour in order to speed up the overall efficiency of the optimisation algorithms. The most common heuristics is the **Armijo rule**.

Algorithm 2 Armijo's rule heuristic

- 1: **initialise.** $\lambda = 1, \alpha, \beta, k = 0$
 - 2: **while** $\theta(\lambda) > \theta(0) + \alpha\lambda\theta'(0)$ **do**
 - 3: $\lambda = \lambda * \beta$
 - 4: $k \leftarrow k + 1$
 - 5: **end while**
 - 6: **return** λ
-

All methods follows the general concept of line search reduction:

Theorem 2.1. Line search reduction

Let $\theta : \mathbb{R} \rightarrow \mathbb{R}$ 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 $z \in [\mu, b]$.

More details in *Lecture V*.

2.3 Descent methods

For direction calculation, different strategies have been attempted. Based on the main method used to determine the descent direction, new methods can be explored:

- Coordinate descent: a descent that is aligned to each variable/dimension present in the objective function.
- Gradient: the descent is defined the gradient of the objective function, using the first-order information:

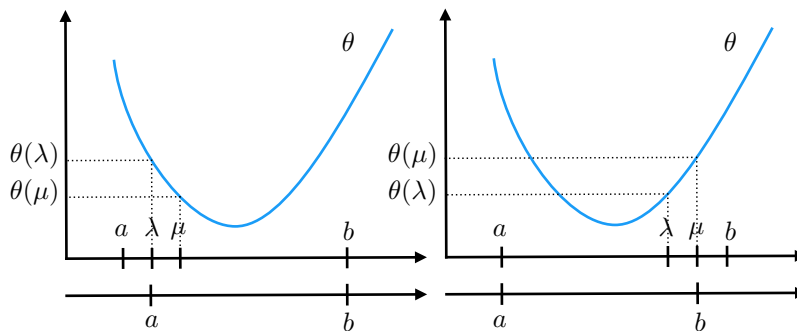


Figure 3: Applying Theorem 2.1 allows to iteratively reduce the search space.

Algorithm 3 Coordinate descent direction

- 1: **for** $j = 1, \dots, n$ **do**
 - 2: $d = \{d_i = 1, \text{ if } i = j; d_i = 0, \text{ if } i \neq j\}$
 - 3: **end for**
-

Algorithm 4 Gradient method direction

- 1: $d = -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}$
-

- Newton: the descent is defined by the gradient of the objective function corrected by the Hessian of the objective, which utilises the second-order information:

Algorithm 5 Newton's method direction

- 1: $d = -H^{-1}(x_k)\nabla f(x_k)$
-

In those methods, the main drawbacks are the simplicity and naivety of the **coordinate direction**, which only allows descent in a single direction at the time; the zigzagging effect on the **gradient method**, which compromises the convergence rate and performance of the algorithm and the challenges caused by inverse matrix in the **Newton's method**. More details in **Lecture V**.

Alternatively, conjugate gradient and quasi-Newton's are derived to circumnavigate those issues. First, with conjugated gradient, it uses the concept of conjugacy to define vector which allows for a better optimisation in each coordinate aided by the gradient value. More details in **Lecture VI**.

Algorithm 6 Conjugate gradient method direction

- 1: $y_0 = x_{k-1}$
 - 2: $d_0 = -\nabla f(y_0)$
 - 3: **for** $j = 1, \dots, n$ **do**
 - 4: $\bar{\lambda}_j = \operatorname{argmin}_{\lambda \geq 0} f(y_{j-1} + \lambda d_{j-1})$
 - 5: $y_j = y_{j-1} + \bar{\lambda}_j d_{j-1}$
 - 6: $d_j = -\nabla f(y_j) + \alpha_j d_{j-1}$, where $\alpha_j = \frac{\|\nabla f(y_j)\|^2}{\|\nabla f(y_{j-1})\|^2}$.
 - 7: **end for**
-

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 $H^{-1}(\bar{x})$. To compute D_k , we use local curvature information, in the attempt to approximate second-order derivatives. For that, let us define the terms:

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

$$q_k = \nabla f(x_{k+1}) - \nabla f(x_k) = H(x_{k+1} - x_k) = H p_k.$$

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:

$$D_{k+1}Hp_j = p_j, \text{ for } j = 1, \dots, k. \quad (2)$$

This condition guarantees that, at the last iteration, $D_n = H^{-1}$. To see that, first, notice the following from (2).

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

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:

$$D_n = [p_0, \dots, p_{n-1}][q_0, \dots, q_{n-1}]^{-1} = H(x_n) \quad (3)$$

Condition (3) 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.

$$\begin{aligned} D_{k+1}q_k &= p_k \\ D_{k+1}(\nabla f(x_{k+1}) - \nabla f(x_k)) &= x_{k+1} - x_k \\ \nabla f(x_{k+1}) &= \nabla f(x_k) + D_{k+1}^{-1}(x_{k+1} - x_k), \end{aligned} \quad (4)$$

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 (4) 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.

There are two version of this approximation:

- 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 the conditions below, called the (2) and (3). For that, notice that:

$$(1) \quad C^{DFP}q_j = C^{DFP}Hp_j$$

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

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

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.

Algorithm 7 Quasi-Newton (DFP) method

```

1: initialise. tolerance  $\epsilon > 0$ , initial point  $x_0$ ,  $D_0 = I$ , iteration count  $k = 0$ .
2: while  $\|\nabla f(x_k)\| > \epsilon$  do
3:    $d = -D_k \nabla f(x_k)$ 
4:    $\bar{\lambda} = \operatorname{argmin}_{\lambda \in \mathbb{R}} \{f(x_k + \lambda d)\}$ 
5:    $x_{k+1} = x_k + \bar{\lambda} d$ 
6:    $p_k = \bar{\lambda} d$ 
7:    $q_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ 
8:    $D_{k+1} = D_k + \frac{p_k p_k^\top}{p_k^\top q_k} - \frac{D_k q_k q_k^\top D_k}{q_k^\top D_k q_k}$ 
9:    $k = k + 1$ 
10: end while
11: return  $x_k$ .

```

- 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 $v_k = p_k - \left(\frac{1}{\tau_k}\right) D_k q_k$, $\tau_k = \frac{q_k^\top D_k q_k}{p_k^\top q_k}$, 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 (2) and (3). 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 $D_{k+1} = B_{k+1}^{-1} = (B_k + \bar{C}_k^{BFGS})^{-1}$, with:

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

Algorithm 8 Quasi-Newton (BFGS) method

- 1: **initialise.** tolerance $\epsilon > 0$, initial point x_0 , $D_0 = I$, iteration count $k = 0$.
 - 2: **while** $\|\nabla f(x_k)\| > \epsilon$ **do**
 - 3: $d = -D_k \nabla f(x_k)$
 - 4: $\bar{\lambda} = \operatorname{argmin}_{\lambda \in \mathbb{R}} \{f(x_k + \lambda d)\}$
 - 5: $x_{k+1} = x_k + \bar{\lambda} d$
 - 6: $p_k = \bar{\lambda} d$
 - 7: $q_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
 - 8: $D_{k+1} = D_k + \frac{p_k p_k^\top}{p_k^\top q_k} \left(1 + \frac{q_k^\top D_k q_k}{p_k^\top q_k}\right) - \frac{D_k q_k p_k^\top + p_k q_k^\top D_k}{p_k^\top q_k}$
 - 9: $k = k + 1$
 - 10: **end while**
 - 11: **return** x_k .
-