Contents

1 Non-linear Programming Problems

1.1 Basic Terminology and Notations

Definition 1.1.1 ▸ General Non-linear Programming (NLP) Problems

Define the function $f : \mathbf{R}^n \to \mathbf{R}$. Let $\boldsymbol{x} \in \mathbf{R}^n$ be a vector, then a general NLP problem aims to **optimise** (i.e. maximise or minimise) $f(\boldsymbol{x})$ subject to the constraint $\boldsymbol{x} \in S \subseteq \mathbf{R}^n$, where

- f is known as the **objective function**;
- S is known as the **feasible set**;
- A solution (point) $\boldsymbol{x} \in S$ is known as a **feasible solution (point)**. Otherwise, it is known as an **infeasible solution(point)**.

Remark. Note that to maximise $f(\mathbf{x})$ is equivalent to minimising $-f(\mathbf{x})$, so it suffices to only study minimisation problems.

The word "optimal", however, can be ambiguous due to its qualitative nature. Thus, we shall define what it means to be optimal quantitatively with more rigorous terms.

Definition 1.1.2 ▸ Optimal Solution

Consider a minimisation problem subject to constraint $\boldsymbol{x} \in S \subseteq \boldsymbol{R}^n$ whose objective function is $f(\boldsymbol{x})$. A feasible solution $\boldsymbol{x}^*$ is called an **optimal solution** if $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in S$. We can write

$$\boldsymbol{x}^* = \underset{\boldsymbol{x} \in S}{\operatorname{argmin}} f(\boldsymbol{x}).$$

$f(\boldsymbol{x}^*)$ is then known as the **optimal value**.

Remark. For maximisation problems, we can write

$$\boldsymbol{x}^* = \underset{\boldsymbol{x} \in S}{\operatorname{argmax}} f(\boldsymbol{x})$$

Note that not all optimisation problems have an optimal solution. We shall still expect to encounter problems for which no optimal solution nor value exists.

Definition 1.1.3 ▸ Unboundedness

Consider a minimisation problem subject to constraint $\boldsymbol{x} \in S \subseteq \boldsymbol{R}^n$ whose objective function is $f(\boldsymbol{x})$. The objective value is said to be **unbounded** if for all $K \in \mathbf{R}$, there exists some $\boldsymbol{x} \in S$ such that $f(\boldsymbol{x}) < K$.

1.2 Unconstrained Non-linear Programs

To introduce the notion of an unconstrained NLP, we shall first define the openness of a set.

Definition 1.2.1 ▸ Open Set

Let $S \subseteq \boldsymbol{R}^n$ be a set. S is called **open** if for all $\boldsymbol{x} \in S$ there exists $\epsilon > 0$ such that the ball

$$B(\boldsymbol{x}, \epsilon) \coloneqq \{\boldsymbol{y} \in \boldsymbol{R}^n : \ \|\boldsymbol{y} - \boldsymbol{x}\| < \epsilon\}$$

is a subset of S.

Definition 1.2.2 ▸ Unconstrained NLP

An **unconstrained** NLP is an NLP whose feasible set $\mathcal{X}$ is an **open** subset of $\mathbf{R}^n$.

1.3 Constrained Non-linear Programs

Similarly, to introduce the notion of a constrained NLP, we shall first define the closed-ness of a set.

Definition 1.3.1 ▸ Closed Set

Let $S \subseteq \boldsymbol{R}^n$ be a non-empty set. S is said to be **closed** if for all convergent sequences $\{\boldsymbol{x}_i\}_{i=1}^{\infty}$ with $\boldsymbol{x}_i \in S$ for $i = 1, 2, \cdots$, the limit $\lim_{i \to \infty} \boldsymbol{x}_i \in S$.

The empty set and Euclidean spaces $\mathbb{R}^n$ are both open and closed.

> *Remark.* Note that a set which is not open may not necessarily be closed. However, a set is open if and only if its complement is closed.

Theorem 1.3.2 ▸ Intersection of Closed Sets

If C_1 and C_2 are both closed, then $C_1 \cap C_2$ is closed.

Proof. The case where $C_1 \cap C_2 = \varnothing$ is trivial. If $C_1 \cap C_2 \neq \varnothing$, let $\{\boldsymbol{x}_i\}_{i=1}^{\infty}$ be an arbitrary convergent sequence in $C_1 \cap C_2$. Since $\{\boldsymbol{x}_i\}_{i=1}^{\infty} \in C_1$ which is closed, we have $\lim_{i\to\infty} \boldsymbol{x}_i \in C_1$. Similarly, $\lim_{i\to\infty} \boldsymbol{x}_i \in C_2$. Therefore, $\lim_{i\to\infty} \boldsymbol{x}_i \in C_1 \cap C_2$.

Therefore, $C_1 \cap C_2$ is closed. □

We then follow up by introducing three important closed sets.

Proposition 1.3.3

Let $g : \mathbf{R}^n \to \mathbf{R}$ be a continuous function, then the sets

$$S_1 = \{\boldsymbol{x} \in \mathbf{R}^n : g(\boldsymbol{x}) \leq 0\},$$
$$S_2 = \{\boldsymbol{x} \in \mathbf{R}^n : g(\boldsymbol{x}) \geq 0\},$$
$$S_3 = \{\boldsymbol{x} \in \mathbf{R}^n : g(\boldsymbol{x}) = 0\}$$

are closed.

Proof. Consider S_1. Let $\{\boldsymbol{x}_i\}_{i=1}^{\infty}$ be any convergent sequence with $\boldsymbol{x}_i \in S_1$ for $i = 1, 2, \cdots$, then

$$g\left(\lim_{i\to\infty} \boldsymbol{x}_i\right) \leq 0$$

since $\boldsymbol{x}_i \leq 0$. Therefore, $\lim_{i\to\infty} \boldsymbol{x}_i \in S_1$ and so S_1 is closed.
S_2 and S_3 can be proved similarly. □

By Theorem 1.3.2, we know that $S_1 \cup S_2 \cup S_3$ is closed, which motivates the following definition:

Definition 1.3.4 ▸ Constrained NLP

A **constrained** NLP is an NLP whose feasible set

$$S \coloneqq \left\{\boldsymbol{x} \in \mathbf{R}^n : g_i(\boldsymbol{x}) = 0, i = 1, 2, \cdots, p, h_j(\boldsymbol{x}) \leq 0, j = 1, 2, \cdots, q\right\}$$

is **closed**, where each of the g_i's is known as an equality constraint and each of the h_j's is known as an inequality constraint.

If the feasible set of a constraind NLP is *bounded*, then it will be easy to find an optimal solution.

Definition 1.3.5 ▶ Boundedness

Let $S \subseteq \mathbf{R}^n$. S is said to be **bounded** if there exists some $M \in \mathbb{R}^+$ such that $\|\boldsymbol{x}\| \leq M$ for all $\boldsymbol{x} \in S$.

Note that a closed set may not be bounded and a bounded set may not be closed, so we are motivated to define a *closed and bounded set* rigorously.

Definition 1.3.6 ▶ Compact Set

A set $S \subseteq \mathbf{R}^n$ is **compact** if it is closed and bounded.

It turns out that, on a compact feasible set, the existence of global minimiser is guaranteed!

Theorem 1.3.7 ▶ Weiestrass Theorem

Let f be a continuous function on some non-empty $S \subseteq \mathbf{R}^n$. If S is compact, then f has a global minimiser and a global maximiser.

2 Convex Functions

2.1 Convexity of Sets and Functions

Intuitively, we describe two types of shapes in natural languages: the shapes which, if you choose any of its edges, lies in the same side of that edge, and the shapes which span across both sides from some chosen edge of its.

Graphically, this means that some shapes are "convex" to all directions, where as some other shapes are "concave". We shall define this rigorously as follows:

Definition 2.1.1 ▸ Convex Set

A set $D \subseteq \mathbf{R}^n$ is said to be **convex** if for all $\boldsymbol{x}, \boldsymbol{y} \in D$ and for all $\lambda \in [0,1]$,

$$\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in D.$$

We can define convexity over functions as follows:

Definition 2.1.2 ▸ Convex Function

A function $f : D \to \mathbf{R}^n$ is said to be **convex** if for all $\boldsymbol{x}, \boldsymbol{y} \in D$ and for all $\lambda \in [0,1]$,

$$f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}).$$

Definition 2.1.3 ▸ Concave Function

A function $f : D \to \mathbf{R}^n$ is said to be **concave** if for all $\boldsymbol{x}, \boldsymbol{y} \in D$ and for all $\lambda \in [0,1]$,

$$f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \geq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}).$$

Remark. A function which is not convex must be concave. However, a function which is convex may not be non-concave (consider $f(x) = x$).

We may derive the following relationship between a convex set and a convex function:

Proposition 2.1.4 ▸ Relations between Convex Sets and Convex Functions

Let $D \subseteq \mathbf{R}^n$ be a convex set and let $f : D \to \mathbf{R}$ be a convex function, then for all $\alpha \in \mathbf{R}$, the set

$$S_\alpha := \{\boldsymbol{x} \in D : f(\boldsymbol{x}) \leq \alpha\}$$

is convex.

Proof. Take $\boldsymbol{x}, \boldsymbol{y} \in S_\alpha$, then $f(\boldsymbol{x}) \leq \alpha$ and $f(\boldsymbol{y}) \leq \alpha$. Note that for any $\lambda \in [0,1]$, we have

$$\begin{aligned} f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) &\leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}) \\ &\leq \lambda\alpha + (1-\lambda)\alpha \\ &= \alpha. \end{aligned}$$

Therefore, $\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in S_\alpha$, and so S_α is convex. □

Next, we introduce the notion of an *epigraph*.

Definition 2.1.5 ▸ Epigraph

Let $f : D \to \mathbf{R}$ be a function over a convex set $D \subseteq \mathbf{R}^n$. The **epigraph** of f is the set $E_f \subseteq \mathbf{R}^{n+1}$ defined by

$$E_f := \{(\boldsymbol{x}, \alpha) : \boldsymbol{x} \in D, \alpha \in \mathbf{R}, f(\boldsymbol{x}) \leq \alpha\}.$$

Remark. A trivial result: $D \times \text{range}(f) \subseteq E_f$.

Note that graphically, the epigraph of a function is just the region above the graph of the function.

Proposition 2.1.6 ▸ Convexity of Epigraph

Let $f : D \to \mathbf{R}$ be a function over a convex set $D \subseteq \mathbf{R}^n$. The epigraph E_f is convex if and only if f is convex.

Proof. Suppose E_f is convex. Take any $\boldsymbol{x}, \boldsymbol{y} \in D$, then $(\boldsymbol{x}, f(\boldsymbol{x})), (\boldsymbol{y}, f(\boldsymbol{y})) \in E_f$. Let $\lambda \in [0,1]$, we have

$$\lambda(\boldsymbol{x}, f(\boldsymbol{x})) + (1-\lambda)(\boldsymbol{y}, f(\boldsymbol{y})) = (\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}, \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y})) \in E_f.$$

Therefore,

$$f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}),$$

and so f is convex.

Suppose conversely that f is convex. For any $\boldsymbol{x}, \boldsymbol{y} \in D$ and any $\alpha, \beta \in \mathbf{R}$ such that $f(\boldsymbol{x}) \leq \alpha$ and $f(\boldsymbol{y}) \leq \beta$, we have $(\boldsymbol{x}, \alpha), (\boldsymbol{y}, \beta) \in E_f$. For all $\lambda \in [0,1]$, consider

$$\lambda(\boldsymbol{x}, \alpha) + (1-\lambda)(\boldsymbol{y}, \beta) = (\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}, \lambda\alpha + (1-\lambda)\beta).$$

Since f is convex, we have

$$\begin{aligned} f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) &\leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}) \\ &\leq \lambda\alpha + (1-\lambda)\beta \end{aligned}$$

for all $\lambda \in [0,1]$. Therefore, $(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}, \lambda\alpha + (1-\lambda)\beta) \in E_f$, and so E_f is convex. □

Lastly, we generalise the notion of a *convex combination*.

Proposition 2.1.7 ▸ Generalised Convex Combination

Let $k \in \mathbb{N}^+$ and let $f : S \to \mathbf{R}$ be a convex function on the convex set $S \subseteq \mathbf{R}^n$ and let $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_k \in S$, then

$$f\left(\sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i\right) \leq \sum_{i=1}^{k} \lambda_i f(\boldsymbol{x}_i),$$

where $\sum_{i=1}^{k} \lambda_i = 1$ and $\lambda_i \geq 0$ for $i = 1, 2, \cdots, k$.

Proof. The case where $k = 1$ is trivial.

Suppose that there exists some $n \in \mathbb{N}^+$ such that

$$f\left(\sum_{i=1}^{n} \lambda_i \boldsymbol{x}_i\right) \leq \sum_{i=1}^{n} \lambda_i f(\boldsymbol{x}_i),$$

□

2.2 Tangent Plane Characterisation

Recall that for a function $f : \mathbf{R}^n \to \mathbf{R}$, the *gradient vector* of f at $\mathbf{x}$ is given by

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(\boldsymbol{x}) \\ \frac{\partial}{\partial x_2} f(\boldsymbol{x}) \\ \vdots \\ \frac{\partial}{\partial x_n} f(\boldsymbol{x}) \end{bmatrix}.$$

Proposition 2.2.1 ▸ Directional Derivative

Let f be a function over $D \subseteq \mathbf{R}^n$ and let $\boldsymbol{d} \in \mathbf{R}^n$ be non-zero, then the directional derivative of f at $\boldsymbol{x}$ along $\boldsymbol{d}$ is

$$\nabla f(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{d} = \lim_{\lambda \to 0} \frac{f(\boldsymbol{x} + \lambda \boldsymbol{d}) - f(\boldsymbol{x})}{\lambda}.$$

In particular, if $\|\mathbf{d}\| = 1$, then the above gives the rate of change of f in the direction of $\boldsymbol{d}$.

Remark. f increases the fastest along the direction of $\nabla f(\boldsymbol{x})$ and decreases the fastest along the direction of $-\nabla f(\boldsymbol{x})$ for any $\boldsymbol{x} \in \mathbf{R}^n$.

The gradient vector of a function allows us to establish its tangent plane at a given point. Intuitively, we can see that a function is convex if its tangent plane lies below its graph. This can be described rigorously as follows:

Proposition 2.2.2 ▸ Tangent Plane Characterisation of Convex Functions

Let f be a function over an open convex set $S \subseteq \mathbf{R}^n$ with continuous first partial derivatives, then f is convex if and only if

$$f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}} (\boldsymbol{y} - \boldsymbol{x}) \leq f(\boldsymbol{y})$$

for all $\boldsymbol{x}, \boldsymbol{y} \in S$. In particular, f is strictly convex if and only if the above inequality is strict.

Proof. Suppose that f is convex. Let $\boldsymbol{x}, \boldsymbol{y} \in S$ and let $\lambda \in [0, 1]$, then we have

$$f\left(\boldsymbol{x} + \lambda(\boldsymbol{y} - \boldsymbol{x})\right) = f\left(\lambda \boldsymbol{y} + (1 - \lambda)\boldsymbol{x}\right) \leq \lambda f(\boldsymbol{y}) + (1 - \lambda) f(\boldsymbol{x}).$$

Therefore,

$$\frac{f(\boldsymbol{x}+\lambda(\boldsymbol{y}-\boldsymbol{x}))-f(\boldsymbol{x})}{\lambda} \leq f(\boldsymbol{y})-f(\boldsymbol{x}).$$

Taking the limit as $\lambda \to 0$ on both sides, we have

$$\nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{x}) = \lim_{\lambda\to 0} \frac{f(\boldsymbol{x}+\lambda(\boldsymbol{y}-\boldsymbol{x}))-f(\boldsymbol{x})}{\lambda} \leq f(\boldsymbol{y})-f(\boldsymbol{x}),$$

and so

$$f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{x}) \leq f(\boldsymbol{y}).$$

Suppose conversely that

$$f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{x}) \leq f(\boldsymbol{y}).$$

Take $\boldsymbol{u}, \boldsymbol{v} \in S$ and let $\boldsymbol{w} = \lambda\boldsymbol{u} + (1-\lambda)\boldsymbol{v}$ for some $\lambda \in [0,1]$, then

$$\boldsymbol{u}-\boldsymbol{w} = (1-\lambda)(\boldsymbol{u}-\boldsymbol{v}), \qquad \boldsymbol{v}-\boldsymbol{w} = -\lambda(\boldsymbol{u}-\boldsymbol{v}).$$

Therefore, we have

$$\begin{aligned} f(\boldsymbol{w}) + \nabla f(\boldsymbol{w})^{\mathrm{T}}(\boldsymbol{u}-\boldsymbol{w}) &\leq f(\boldsymbol{u}), \\ f(\boldsymbol{w}) + \nabla f(\boldsymbol{w})^{\mathrm{T}}(\boldsymbol{v}-\boldsymbol{w}) &\leq f(\boldsymbol{v}). \end{aligned}$$

Note that

$$\lambda\big(f(\boldsymbol{w}) + \nabla f(\boldsymbol{w})^{\mathrm{T}}(\boldsymbol{u}-\boldsymbol{w})\big) + (1-\lambda)\big(f(\boldsymbol{w}) + \nabla f(\boldsymbol{w})^{\mathrm{T}}(\boldsymbol{v}-\boldsymbol{w})\big) = f(\boldsymbol{w}),$$

so we have

$$\begin{aligned} f(\lambda\boldsymbol{u} + (1-\lambda)\boldsymbol{v}) &= f(\boldsymbol{w}) \\ &\leq \lambda f(\boldsymbol{u}) + (1-\lambda) f(\boldsymbol{v}). \end{aligned}$$

Hence, f is convex.

We can similarly prove that f is strictly convex if $f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{x}) < f(\boldsymbol{y})$. Now suppose that f is strictly convex but there exists $\boldsymbol{x}, \boldsymbol{y} \in S$ with $\boldsymbol{x} \neq \boldsymbol{y}$ such that

$$f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{x}) \geq f(\boldsymbol{y}).$$

Take $\boldsymbol{z} = \frac{1}{2}\boldsymbol{x} + \frac{1}{2}\boldsymbol{y}$. We have

$$\begin{aligned}\frac{1}{2}f(\boldsymbol{x}) + \frac{1}{2}f(\boldsymbol{y}) &> f(\boldsymbol{z}) \\ &\geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{z} - \boldsymbol{x}) \\ &= f(\boldsymbol{x}) + \frac{1}{2}\nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}) \\ &\geq f(\boldsymbol{x}) + \frac{1}{2}(f(\boldsymbol{y}) - f(\boldsymbol{x})) \\ &= \frac{1}{2}f(\boldsymbol{x}) + \frac{1}{2}f(\boldsymbol{y}),\end{aligned}$$

which is a contradiction. □

Proposition 2.2.2 helps us determine whether a point is the global minimiser of a certain convex function.

Theorem 2.2.3 ▸ Global Minimiser of Convex Functions

Let $f : C \to \mathbf{R}$ be a convex and continuously differentiable function over a convex set $C \subseteq \mathbf{R}^n$. Then $\boldsymbol{x}^* \in C$ is a global minimiser for the minimisation problem

$$\min\{f(\boldsymbol{x}) : \boldsymbol{x} \in C\}$$

if and only if

$$\nabla f(\boldsymbol{x}^*)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^*) \geq 0$$

for all $\boldsymbol{x} \in C$.

Proof. Suppose $\nabla f(\boldsymbol{x}^*)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^*) \geq 0$. By Proposition 2.2.2, we have

$$\nabla f(\boldsymbol{x}^*)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^*) \leq f(\boldsymbol{x}) - f(\boldsymbol{x}^*).$$

This means that $f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)$ for all $\boldsymbol{x} \in C$, and so $\boldsymbol{x}^*$ is a global minimiser.

Suppose that $\boldsymbol{x}^*$ is a global minimiser for f but there exists some $\boldsymbol{x}_0 \in C$ such that

$$\nabla f(\boldsymbol{x}^*)^{\mathrm{T}}(\boldsymbol{x}_0 - \boldsymbol{x}^*) < 0.$$

Consider $\boldsymbol{y} = \lambda\boldsymbol{x}_0 + (1 - \lambda)\boldsymbol{x}^*$ for $\lambda \in (0, 1)$. Notice that $f(\boldsymbol{y}) \geq f(\boldsymbol{x}^*)$, so by using

Proposition 2.2.2 we have

$$\begin{aligned} 0 &\leq f(\boldsymbol{y}) - f(\boldsymbol{x}^*) \\ &\leq -\nabla f(\boldsymbol{y})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}^*) \\ &\leq \nabla f(\boldsymbol{y})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}^*) \\ &= \lambda \nabla f\left(\lambda \boldsymbol{x}_0 + (1-\lambda)\boldsymbol{x}^*\right)^{\mathrm{T}}(\boldsymbol{x}_0 - \boldsymbol{x}^*) \\ &\leq \nabla f\left(\lambda \boldsymbol{x}_0 + (1-\lambda)\boldsymbol{x}^*\right)^{\mathrm{T}}(\boldsymbol{x}_0 - \boldsymbol{x}^*). \end{aligned}$$

Therefore,

$$\begin{aligned} \lim_{\lambda \to 0^+} \left[\nabla f\left(\lambda \boldsymbol{x}_0 + (1-\lambda)\boldsymbol{x}^*\right)^{\mathrm{T}}(\boldsymbol{x}_0 - \boldsymbol{x}^*)\right] &= \nabla f(\boldsymbol{x}^*)^{\mathrm{T}}(\boldsymbol{x}_0 - \boldsymbol{x}^*) \\ &\geq 0, \end{aligned}$$

which is a contradiction. □

2.3 Hessian Matrices

So far we have learnt how to determine a function's convexity by either Definition 2.1.2 or Proposition 2.2.2. However, there are certain functions which are difficult to manipulate and apply those methods algebraically. Therefore, we introduce another method to determine convexity by *Hessian Matrices*.

Definition 2.3.1 ▸ Hessian Matrix

Let $f : S \to \mathbf{R}$ where $S \subseteq \mathbf{R}^n$ is non-empty and let $\boldsymbol{x}$ be an **interior point** of S. The **Hessian** of f at $\boldsymbol{x}$ is the $n \times n$ matrix

$$H_f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1}(\boldsymbol{x}) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(\boldsymbol{x}) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(\boldsymbol{x}) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(\boldsymbol{x}) & \frac{\partial^2 f}{\partial x_2 \partial x_2}(\boldsymbol{x}) & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(\boldsymbol{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(\boldsymbol{x}) & \frac{\partial^2 f}{\partial x_n \partial x_2}(\boldsymbol{x}) & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n}(\boldsymbol{x}) \end{bmatrix}$$

Remark. If f has continuous second order derivatives, then $H_f(\boldsymbol{x})$ is symmetric.

To use Hessian Matrices in convexity test, we also need to introduce the following notion of *(semi)definiteness*:

Definition 2.3.2 ▶ (Semi)Definiteness

et $\boldsymbol{A}$ be a real square matrix.

- $\boldsymbol{A}$ is said to be **positive semidefinite** if $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} \geq 0$ for all $\boldsymbol{x} \in \boldsymbol{R}$.
- $\boldsymbol{A}$ is said to be **positive definite** if $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} > 0$ for all $\boldsymbol{x} \in \boldsymbol{R}$.
- $\boldsymbol{A}$ is said to be **negative semidefinite** if $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} \leq 0$ for all $\boldsymbol{x} \in \boldsymbol{R}$.
- $\boldsymbol{A}$ is said to be **negative definite** if $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} < 0$ for all $\boldsymbol{x} \in \boldsymbol{R}$.
- $\boldsymbol{A}$ is said to be **indefinite** if it is neither positive semidefinite nor negative semidefinite.

To determine the definiteness of a matrix, we may use the following eigenvalue tests:

Theorem 2.3.3 ▶ Eigenvalue Test

If $\boldsymbol{A}$ is a symmetric real square matrix, then:

- *$\boldsymbol{A}$ is positive semidefinite if and only if all eigenvalues of $\boldsymbol{A}$ are non-negative.*
- *$\boldsymbol{A}$ is positive definite if and only if all eigenvalues of $\boldsymbol{A}$ are positive.*
- *$\boldsymbol{A}$ is negative semiefinite if and only if all eigenvalues of $\boldsymbol{A}$ are non-positive.*
- *$\boldsymbol{A}$ is negative definite if and only if all eigenvalues of $\boldsymbol{A}$ are negative.*
- *$\boldsymbol{A}$ is indefinite if and only if it has at least one positive eigenvalue and at least one negative eigenvalue.*

Proof. We will only prove that $\boldsymbol{A}$ is positive semidefinite if and only if all eigenvalues of $\boldsymbol{A}$ are non-negative. The rest of the tests can be proven similarly.

Suppose $\boldsymbol{A}$ is positive semidefinite. Let λ be any eigenvalue of $\boldsymbol{A}$ and let $\boldsymbol{x}$ be a corresponding eigenvector, then $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$. Therefore,

$$\lambda\|\boldsymbol{x}\|^2 = \lambda\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} \geq 0.$$

Since $\|\boldsymbol{x}\|^2 \geq 0$, this implies that $\lambda \geq 0$.

Suppose conversely that all eigenvalues of $\boldsymbol{A}$ are non-negative. Consider the diagonal matrix

$$\boldsymbol{D} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}$$

where λ_i is the i-th eigenvalue of $\boldsymbol{A}$. Then there exists some orthogonal square matrix

$\boldsymbol{Q}$ such that

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^{\mathrm{T}}.$$

Let $\boldsymbol{x} \in \mathbf{R}^n$ be an arbitrary vector, then

$$\begin{aligned}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{x} \\ &= \left(\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{x}\right)^{\mathrm{T}} \boldsymbol{D}\left(\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{x}\right) \\ &= \sum_{i=1}^{n}\left(\lambda_i \left\|\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{x}\right\|^2\right) \\ &\geq 0.\end{aligned}$$

Therefore, $\boldsymbol{A}$ is positive semidefinite. □

However, eigenvalues can be troublesome to compute. Thus for **small matrices**, we may use the following test with the *principal minors*:

Definition 2.3.4 ▸ Principal Minor

Let $\boldsymbol{A}$ be an $n \times n$ matrix. The k-th **principal minor** Δ_k of $\boldsymbol{A}$ is defined to be the determinant of the kth principal submatrix of $\boldsymbol{A}$, i.e.,

$$\Delta_k = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{vmatrix}$$

Theorem 2.3.5 ▸ Definiteness Test Using Principal Minors

Let $\boldsymbol{A}$ be a symmetric $n \times n$ matrix, then $\boldsymbol{A}$ is positive definite if and only if $\Delta_k > 0$, and negative definite if and only if $(-1)^k\Delta_k > 0$, for all $k = 1, 2, \cdots, n$.

2.3.1 Taylor's Theorem

Note that we have not given the proof to Theorem 2.3.7. Now to prove it, we need to first introduce *Taylor's Theorem*.

Theorem 2.3.6 ▸ Taylor's Theorem

Suppose that $f : S \to \mathbf{R}$ is a function with continuous second partial derivatives. Consider the set

$$[\boldsymbol{x}, \boldsymbol{y}] \coloneqq \{\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y} : \boldsymbol{x}, \boldsymbol{y} \in \mathbf{R}^n, \lambda \in [0,1]\}.$$

If $[\boldsymbol{x}, \boldsymbol{y}]$ is contained in the interior of S, then there exists some $\boldsymbol{z} \in [\boldsymbol{x}, \boldsymbol{y}]$ such that

$$f(\boldsymbol{y}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}) + \frac{1}{2}(\boldsymbol{y} - \boldsymbol{x})^{\mathrm{T}} H_f(\boldsymbol{z})(\boldsymbol{y} - \boldsymbol{x}).$$

Now, we are able to apply the following tests:

Theorem 2.3.7 ▸ Convexity Test for Differentiable Functions

Let a function f have continuous second order derivatives on an open convex set $D \subseteq \mathbf{R}^n$.

- *$H_f(\boldsymbol{x})$ is positive semidefinite for all $\boldsymbol{x} \in D \iff f$ is convex on D.*
- *$H_f(\boldsymbol{x})$ is positive definite for all $\boldsymbol{x} \in D \implies f$ is strictly convex on D.*
- *$H_f(\boldsymbol{x})$ is negative semidefinite for all $\boldsymbol{x} \in D \iff f$ is concave on D.*
- *$H_f(\boldsymbol{x})$ is negative definite for all $\boldsymbol{x} \in D \implies f$ is strictly concave on D.*
- *$H_f(\boldsymbol{x})$ is indefinite for some $\boldsymbol{x} \in D \implies f$ is neither convex nor concave on D.*

3

Unconstrained Optimisation

3.1 Coercive Functions

Definition 3.1.1 ▶ Coercive Function

A **continuous** function $f : \mathbf{R}^n \to \mathbf{R}$ is said to be **coercive** if

$$\lim_{\|\boldsymbol{x}\| \to \infty} f(\boldsymbol{x}) = +\infty.$$

More formally, f is coercive if and only if for all $M > 0$, there is some $r > 0$ such that $f(\boldsymbol{x}) > M$ whenever $\|\boldsymbol{x}\| > r$.

Theorem 3.1.2 ▶ Global Minimiser of Coercive Functions

If a function $f : \mathbf{R}^n \to \mathbf{R}$ is coercive, then f has at least one global minimiser.

Proof. Take $M = |f(0)| + 1 > 0$. Since f is coercive, there exists some $r > 0$ such that $f(\boldsymbol{x}) > M > f(0)$ whenever $\|\boldsymbol{x}\| > r$. Consider

$$B(0, r) = \{\boldsymbol{x} \in \mathbf{R}^n : \|\boldsymbol{x}\| \leq r\}$$

which is a compact set. So by Weiestrass' Theorem, there exists some $\boldsymbol{x}^* \in B(0, r)$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}) \leq f(0)$ for all $\boldsymbol{x} \in B(0, r)$. Now, this means that for all $\boldsymbol{x} \in \mathbf{R}^n$, we have $(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$, which means that $\boldsymbol{x}^*$ is a global minimiser for f. □

3.2 Critical Points

Definition 3.2.1 ▶ Stationary (Critical) Point

Let $X \subseteq \mathbf{R}^n$ be an open set and let $f : X \to \mathbf{R}$ be a function. An interior point $\boldsymbol{x}^*$ is called a **stationary point** of f if $\nabla f(\boldsymbol{x}^*) = 0$.

Definition 3.2.2 ▶ Saddle Point

A stationary point $\boldsymbol{x}^*$ of a function f which is neither a local minimisr nor a local maximiser is called a **saddle point**.

Corollary 3.2.3

Let $\boldsymbol{x}^*$ be a stationary point of f. If $H_f(\boldsymbol{x}^*)$ is indefinite, then $\boldsymbol{x}^*$ is a saddle point.

4

Numerical Methods

4.1 One-Dimensional Methods

4.1.1 Bisection

Recall the *Intermediate Value Theorem*:

Theorem 4.1.1 ▶ Intermediate Value Theorem

Let f be a continuous function on $[a,b]$ with $f(a)f(b) < 0$, then there exists some $r \in (a,b)$ such that $f(r) = 0$.

Note that $f'(x) = 0$ if x is a local optimiser for f. The bisection method aims to find the solution to $f'(x) = 0$ to determine an approximation for the local optimiser of f.

Technique 4.1.2 ▶ Bisection Search Algorithm

Let f be a one-dimensional function such that $f(a)f(b) < 0$. We initialise bisection search with the interval $[a_1, b_1] = [a, b]$ and a tolerance level $\epsilon > 0$.

At the k-th iteration, set $x_k = \frac{a_k + b_k}{2}$ as the current approximation.

If $b_k - a_k \leq 2\epsilon$ or $f(x_k) = 0$, we can stop the search and output x_k as an approximated root of $f(x) = 0$.

Else, we update the interval by

$$[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, x_k], & \text{if } f(a_k)f(x_k) < 0 \\ [x_k, b_k], & \text{if } f(b_k)f(x_k) < 0 \end{cases}.$$

Let $x^* \in [a,b]$ be the true root of $f(x) = 0$. Note that if the algorithm terminates at the k-th iteration, it means that $|b_k - a_k| \leq 2\epsilon$. Since $x^* \in [a_k, b_k]$ and $x_k = \frac{b_k + a_k}{2}$, we have

$$|x_k - x^*| \leq \frac{|b_k - a_k|}{2} \leq \epsilon.$$

Furthermore, notice that the size of the interval $[a_i, b_i]$ is halved after each iteration, so we

have $|b_k - a_k| = \frac{|b_1 - a_1|}{2^{k-1}}$. This means to have $|b_k - a_k| \leq 2\epsilon$, we need at least

$$k = \left\lceil \log_2 \frac{|b_1 - a_1|}{\epsilon} \right\rceil$$

iterations. In other words, bisection search with tolerance level ϵ and initial interval width n is an $\mathcal{O}\left(\log \frac{n}{\epsilon}\right)$ algorithm.

Code Snippet 4.1.3 ▸ Bisection Search Algorithm in Matlab

```
function [x, k] = bisection(f, a, b, t)
    fa = feval(f, a);
    fb = feval(f, b);
    if (sign(fa) == sign(fb))
        error("f(a), f(b) have same sign.");
    end
    k = 1;
    while abs(b - a) > t
        x = (a + b) / 2;
        fx = feval(f, x);
        if (fx == 0)
            break;
        end
        if (sign(fx) == sign(fa))
            a = x;
            fa = fx;
        else
            b = x;
            fb = fx;
        end
        k += 1;
    end
end
```

4.1.2 Newton's Method

Let $f(x)$ be a function, then by Taylor expansion we can approximate f by

$$f(x) \approx q(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2} f''(x_k)(x - x_k)^2,$$

where x_k is some approximation of the global minimiser of f. Let x^* be the global minimiser for q, then

$$x^* = x_k - \frac{f'(x_k)}{f''(x_k)}.$$

This motivates the following algorithm:

Technique 4.1.4 ▸ Newton's Method Algorithm

Let f be a one-dimensional function. We initialise Newton's method with an initial approximation x_0 and a tolerance level $\epsilon > 0$.

At the k-th iteration, set $x_k = x_{k-1} - \frac{f'(x_k)}{f''(x_k)}$. If $|f'(x_k)| \leq \epsilon$, we can stop the algorithm and output x_k as an approximated minimiser of f.

Code Snippet 4.1.5 ▸ Newton's Method Algorithm in Matlab

```
function [x, k] = newton(df, ddf, x0, t)
    k = 0;
    x = x0;
    while abs(feval(df, x)) > t
        x -= feval(df, x) / feval(ddf, x);
        k += 1;
    end
end
```

The rate of convergence of Newton's method is generally very fast, but for non-convex functions, there is a risk that the algorithm fails to find the approximation.

4.1.3 Golden Section Method

In some cases, we may encounter a function which is not differentiable (and thus we cannot use Newton's method to our advantage). It turns out, however, that if the function is *unimodal*, then we can easily approximate its global minimiser.

Definition 4.1.6 ▸ Unimodal Function

A function f is said to be **unimodal** on $[a, b]$ if it has exactly one global minimiser (maximiser) x^* in $[a, b]$ and is strictly decreasing (increasing) on $[a, x^*]$ and strictly increasing (decreasing) on $[x^*, b]$.

Technique 4.1.7 ▸ Golden Section Method

Let f be a unimodal function. Consider $[a_0, b_0]$ as the initial interval. For each iteration, we consider

$$\lambda_n = b_n - \Phi(b_n - a_n)$$
$$\mu_n = a_n + \Phi(b_n - a_n)$$

where $\Phi = \frac{\sqrt{5}-1}{2}$ is the Golden Ratio constant. Then we update the interval by:

$$[a_{n+1}, b_{n+1}] = \begin{cases} [\lambda_n, b_n], & \text{if } f(\lambda_n) > f(\mu_n) \\ [a_n, \mu_n], & \text{if } f(\lambda_n) < f(\mu_n) \\ [\lambda_n, \mu_n], & \text{if } f(\lambda_n) = f(\mu_n) \end{cases}.$$

The reason why we use Φ to generate the intervals recursively is because that it reduces the number of computations. Consider that $[a_k, b_k] = [\lambda_{k-1}, b_{k-1}]$, then

$$\begin{aligned} \lambda_k &= b_k - \Phi(b_k - a_k) \\ &= b_{k-1} - \Phi(b_{k-1} - \lambda_{k-1}) \\ &= b_{k-1} - \Phi(b_{k-1} - b_{k-1} + \Phi(b_{k-1} - a_{k-1})) \\ &= b_{k-1} - \Phi^2(b_{k-1} - a_{k-1}) \\ &= b_{k-1} - (1 - \Phi)(b_{k-1} - a_{k-1}) \\ &= a_{k-1} + \Phi(b_{k-1} - a_{k-1}) \\ &= \mu_{k-1}. \end{aligned}$$

However, note that we have already computed μ_{k-1} in the previous iteration! Therefore, by applying ideas of *dynamic programming*, this eliminates the need to re-compute its value. Similarly, we can show that $\mu_k = \lambda_{k-1}$ if $f(\lambda_k) < f(\mu_k)$.

Code Snippet 4.1.8 ▸ Golden Section Algorithm in Matlab

```
function [lhs, rhs, k] = golden_section(f, a, b, t)
    lhs = a;
    rhs = b;
    k = 0;
    phi = (sqrt(5) - 1) / 2;
    lambda = b - phi * (b - a);
    mu = a + phi * (b - a);
```

```
    while abs(b - a) > t
        f_lambda = feval(f, lambda);
        f_mu = feval(f, mu);
        if (f_lambda > f_mu)
            a = lambda;
            lambda = mu;
        elseif (f_lambda < f_mu)
            b = mu;
            mu = lambda;
        else
            a = lambda;
            b = mu;
            lambda = b - phi * (b - a);
            mu = a + phi * (b - a);
        end
        lhs = a;
        rhs = b;
        k += 1;
    end
end
```

4.2 Multivariable Methods

Consider the following general framework of optimisation algorithms:

- Initialise with an initial guess $\boldsymbol{x}_0$.
- At each iteration:
  - If $\boldsymbol{x}_k$ is considered optimal, stop the search and output.
  - Else, set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$.

$\boldsymbol{p}_k$ is the search direction and α_k is the step size.

4.2.1 Newton's Method

Similar to the one-dimensional case, let f be a multivariable function with continuous second order partial derivatives. Given $\boldsymbol{x_k} \in \mathbf{R}^n$ as an approximation of the global minimiser

of f, by Taylor expansion there is some quadratic function q such that

$$f(\boldsymbol{x}) \approx q(\boldsymbol{x}) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k)^{\mathrm{T}} H_f(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k).$$

The global minimiser of q is given by

$$\hat{\boldsymbol{x}} = \boldsymbol{x}_k - H_f(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k).$$

However, for the above equation to hold, we require two important assumptions: first, H_f must be non-singular at $\hat{\boldsymbol{x}}$; second, $H_f(\boldsymbol{x})$ is *Lipschitz continuous* in a neighbourhood of $\hat{\boldsymbol{x}}$. To formally introduce *Lipschitz continuity*, we state the following definitions:

Definition 4.2.1 ▸ Frobenius Norm

Let $\boldsymbol{A} \in \mathbf{R}^{n\times n}$ be a square matrix. The **Frobenius norm** of $\boldsymbol{A}$ is defined by

$$\|\boldsymbol{A}\|_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2.$$

We can show that $\|\boldsymbol{A}\boldsymbol{x}\| \leq \|\boldsymbol{A}\|_F\|\boldsymbol{x}\|$ for any vector $\boldsymbol{x} \in \mathbf{R}^n$ and that $\|\boldsymbol{A}\boldsymbol{B}\| \leq \|\boldsymbol{A}\|_F\|\boldsymbol{B}\|_F$ for any $n \times n$ matrix $\boldsymbol{B}$.

Definition 4.2.2 ▸ Lipschitz Continuity

Let $f : \mathbf{R}^n \to \mathbf{R}^{n\times n}$ be a function. f is **Lipschitz continuous** at $\boldsymbol{x}^*$ if there exists some $\delta, L > 0$ such that for all $\boldsymbol{x}, \boldsymbol{y} \in B(\boldsymbol{x}^*, \delta)$,

$$\|f(\boldsymbol{x}) - f(\boldsymbol{y})\|_F \leq L\|\boldsymbol{x} - \boldsymbol{y}\|.$$

In practice when using Newton's method, we choose a neighbourhood of $\hat{\boldsymbol{x}}$ which is small enough by setting $\delta = \frac{1}{\left\|H_f(\hat{\boldsymbol{x}})^{-1}\right\|_F}$.

Technique 4.2.3 ▸ Multi-dimensional Newton's Method Algorithm

Let f be a function. We initialise Newton's method with an initial approximation $\boldsymbol{x}_0$ and a tolerance level $\epsilon > 0$.

At the k-th iteration, set $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} - H_f(\boldsymbol{x}_{k-1})^{-1}\nabla f(\boldsymbol{x}_{k-1})$. If $\|\nabla f(\boldsymbol{x}_k)\| \leq \epsilon$, we can stop the algorithm and output $\boldsymbol{x}_k$ as an approximated minimiser of f.

Note that in the multi-dimensional case, although Newton's method still retain a fast rate of convergence, the computational cost per iteration is very high.

Proposition 4.2.4 ▸ Convergence of the Newton Method

If $\boldsymbol{x}_0$ is sufficiently close to $\boldsymbol{x}^*$, then the sequence $\{\boldsymbol{x}_k\}$ generated by the Newton Method converges to $\boldsymbol{x}^*$ quadratically, i.e., there exists some $M \in \mathbb{R}$ such that

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| \leq M\|\boldsymbol{x}_k - \boldsymbol{x}^*\|^2.$$

Note that the search direction of Newton's method is $-H_f(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k)$ and the step size is 1. However, we can choose a better step size α_k given by

$$\alpha_k = \underset{\alpha \geq 0}{\operatorname{argmin}}\, f(\boldsymbol{x}_{k+1}) = \underset{\alpha \geq 0}{\operatorname{argmin}}\, f\left(\boldsymbol{x}_k - \alpha_k H_f(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k)\right).$$

4.2.2 Armijo Line Search

Let $\boldsymbol{x}_0$ be an approximation of some optimisation algorithm with search direction $\boldsymbol{p}_0$ and step size α. We can obtain the next approximation as

$$\hat{\boldsymbol{x}} = \boldsymbol{x}_0 + \alpha \boldsymbol{p}_0.$$

Correspondingly, the average change in the function value is given by

$$\frac{f(\hat{\boldsymbol{x}}) - f(\boldsymbol{x}_0)}{\alpha} = \frac{f(\boldsymbol{x}_0 + \alpha\boldsymbol{p}_0) - f(\boldsymbol{x}_0)}{\alpha}.$$

Note that the rate of change along $\boldsymbol{p}_0$ is given by $\nabla f(\boldsymbol{x}_0)^{\mathrm{T}}\boldsymbol{p}_0$. Suppose we wish α to be sufficiently small, then we can fix some $\sigma \in (0,1)$ such that

$$\frac{f(\boldsymbol{x}_0 + \alpha\boldsymbol{p}_0) - f(\boldsymbol{x}_0)}{\alpha} \leq \sigma\nabla f(\boldsymbol{x}_0)^{\mathrm{T}}\boldsymbol{p}_0.$$

If the above inequality holds true, it means that the average rate of decrease in the value of f along $\boldsymbol{p}_0$ with step size α is at least σ of the instantaneous rate of decrease. If the choice of α is not ideal enough, we can then try $\beta\alpha$ for some $\beta \in (0,1)$. This leads to the *Armijo line search* algorithm.

Technique 4.2.5 ▸ Armijo Line Search Algorithm

Take $\sigma \in (0, 0.5)$ and $\beta \in (0,1)$. Initialise α_0 with a potentially large value.

At the k-th iteration, consider $\alpha_k = \beta\alpha_{k-1}$. If

$$f(\boldsymbol{x}_0 + \alpha_k\boldsymbol{p}_0) \leq f(\boldsymbol{x}_0) + \sigma\alpha_k\nabla f(\boldsymbol{x}_0)^{\mathrm{T}}\boldsymbol{p}_0,$$

we can output α_k as an sufficiently optimal choice of α.

We can then combine the Armijo line search with Newton's method as follows:

Code Snippet 4.2.6 ▸ Newton's Method with Armijo Line Search in Matlab

```
function [alpha, k] = armijo(f, fx, nabla, x, p, sigma,
    beta, init_a)
    alpha = init_a;
    dec = nabla' * p;
    k = 0;
    while feval(f, x + alpha * p) > sigma * alpha * dec
        alpha *= beta;
        k += 1;
    end
end

function [x, k] = newton_armijo(f, df, ddf, x0, t)
    x = x0;
    i = 0;
    alpha = 2;
    df_val = feval(df, x);
    while norm(df_val) > t
        p = -feval(ddf, x) \ df_val;
        [alpha, i] = armijo(f, feval(f, x), df_val, x, p,
            0.48, 0.95, alpha);
        x += alpha * p;
    end
end
```

4.3 Steepest Descent

Recall that f decreases most rapidly along the direction of $-\nabla f$. So we can take

$$\hat{\boldsymbol{d}} = -\frac{\nabla f(\boldsymbol{x})}{\|\nabla f(\boldsymbol{x})\|}$$

at any $\boldsymbol{x}$, where $-\nabla f(\boldsymbol{x})$ is known as the *steepest descent direction*. Recall 2.2.1, so

$$\begin{aligned} f(\boldsymbol{x}+\delta\hat{\boldsymbol{d}}) &= f(\boldsymbol{x}) + \delta\nabla f(\boldsymbol{x})^{\mathrm{T}}\hat{\boldsymbol{d}} \\ &= f(\boldsymbol{x}) - \delta\frac{\|\nabla f(\boldsymbol{x})\|^2}{\|\nabla f(\boldsymbol{x})\|} \\ &= f(\boldsymbol{x}) - \delta\|\nabla f(\boldsymbol{x})\|. \end{aligned}$$

Technique 4.3.1 ▸ Steepest Descent Method

Let f be an objective function and $\boldsymbol{x}_0$ be an initial guess. Let $\epsilon > 0$ be the tolerance level.

At the k-th iteration, we evaluate the steepest descent direction

$$\boldsymbol{d}_k = -\nabla f\left(\boldsymbol{x}_k\right).$$

If $\|\boldsymbol{d}_k\| < \epsilon$, $\boldsymbol{x}_k$ is an approximate minimiser. Otherwise, compute

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + t_k\boldsymbol{d}_k,$$

where t_k is such that $f(\boldsymbol{x}_k + t_k\boldsymbol{d}_k)$ is minimised and is known as the step size.

Code Snippet 4.3.2 ▸ Steepest Descent Algorithm in Matlab

```
function [x, d, k] = steepest_descent(f, nabla, x0, t)
    x = x0;
    grad = feval(nabla, x);
    k = 0;
    while norm(grad) > t
        d = -grad;
        fx = feval(f, x);
        [step, i] = armijo(f, fx, nabla, x, d, 0.48, 0.95,
         ↪ 2);
        x += step * d;
        grad = feval(nabla, x);
        k += 1;
    end
end
```

Intuitively, suppose $\boldsymbol{x}_0$ is some initial approximation located at the level set $f(\boldsymbol{x}) = k$. Using the steepest descent method, we would search along the direction that is **orthogonal** to the level set at $\boldsymbol{x}_0$. We denote this direction as $\boldsymbol{d}$. Note that we will only find the next approximation when $f(\boldsymbol{x})$ **cannot decrease further**, i.e., we encounter some other level set $f(\boldsymbol{x}) = h$ to which $\boldsymbol{d}$ is a tangential direction.

With a graphical illustration, one might observe that the approximations obtained via the steepest descent method trace out a "zig-zag" path with an orthogonal turn between any two consecutive search directions. This behaviour is rigorously described as follows:

Proposition 4.3.3 ▸ Zig-zag Behaviour of Steepest Descent

If $(\boldsymbol{x}_k)$ be a sequence of approximations for the function $f(\boldsymbol{x})$ using steepest descent method, then

$$(\boldsymbol{x}_{k+2} - \boldsymbol{x}_{k+1})^{\mathrm{T}}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = 0$$

for all $k \in \mathbb{N}$.

4.4 Conjugate Gradient Method

Consider the quadratic function

$$\phi(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}$$

where $\boldsymbol{A}$ is symmetric positive definite. Let $\boldsymbol{x}_0$ be an initial approximation. Suppose we obtain $\boldsymbol{x}_1$ along $\boldsymbol{p}_0$ with exact line search, i.e.,

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \alpha_0 \boldsymbol{p}_0, \qquad \alpha_0 = \operatorname*{argmin}_{\alpha > 0} \phi(\boldsymbol{x}_0 + \alpha \boldsymbol{p}_0).$$

Notice that this means that α_0 is a global minimiser to $g_0(\alpha) = \phi(\boldsymbol{x}_0 + \alpha \boldsymbol{p}_0)$, and so $g_0'(\alpha_0) = 0$. Let $\boldsymbol{y}_0 = \boldsymbol{x}_0 + \alpha \boldsymbol{p}_0$, then

$$\begin{aligned} g_0'(\alpha) &= \frac{\partial \phi}{\partial \boldsymbol{y}_0} \cdot \frac{\partial \boldsymbol{y}_0}{\partial \alpha} \\ &= \nabla \phi(\boldsymbol{x}_0 + \alpha \boldsymbol{p}_0)^{\mathrm{T}} \boldsymbol{p}_0. \end{aligned}$$

This implies that $\nabla \phi(\boldsymbol{x}_1)^{\mathrm{T}} \boldsymbol{p}_0 = 0$. Let $\boldsymbol{x}^*$ be the optimal point, we now wish to find some $\boldsymbol{p}_1$ and α^* such that

$$\boldsymbol{x}_1 + \alpha^* \boldsymbol{p}_1 = \boldsymbol{x}^*.$$

Notice that $\nabla\phi(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$, so $\nabla\phi(\boldsymbol{x}^*) = \boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b} = 0$ and $\nabla\phi(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{p}_0 = (\boldsymbol{A}\boldsymbol{x}_1 - \boldsymbol{b})^{\mathrm{T}}\boldsymbol{p}_0 = 0$, so we have

$$\begin{aligned} 0 &= (\boldsymbol{A}\boldsymbol{x}_1 - \boldsymbol{b})^{\mathrm{T}}\boldsymbol{p}_0 \\ &= (\boldsymbol{A}\boldsymbol{x}_1 - \boldsymbol{A}\boldsymbol{x}^*)^{\mathrm{T}}\boldsymbol{p}_0 \\ &= (\boldsymbol{x}_1 - \boldsymbol{x}^*)^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{p}_0 \\ &= -\alpha^*\boldsymbol{p}_1^{\mathrm{T}}\boldsymbol{A}\boldsymbol{p}_0. \end{aligned}$$

The above concludes that if there is $\boldsymbol{p}_0$, $\boldsymbol{p}_1$ and $\alpha^* \in \mathbf{R}$ such that for some initial approximation $\boldsymbol{x}_0$ of the optimiser $\boldsymbol{x}^*$ to a quadratic function ϕ, we have

$$\begin{aligned} \boldsymbol{x}^* &= \boldsymbol{x}_1 + \alpha^*\boldsymbol{p}_1 \\ &= \boldsymbol{x}_0 + \alpha_0\boldsymbol{p}_0 + \alpha^*\boldsymbol{p}_1 \end{aligned}$$

where $\alpha_0 = \operatorname{argmin}_{\alpha>0}\phi(\boldsymbol{x}_0 + \alpha\boldsymbol{p}_0)$, then necessarily $\boldsymbol{p}_1^{\mathrm{T}}\boldsymbol{A}\boldsymbol{p}_0 = 0$.

Definition 4.4.1 ▸ Conjugate Vectors

A set of vectors $\{\boldsymbol{p}_0, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_n\}$ is **conjugate** with respect to $\boldsymbol{A}$, where $\boldsymbol{A} \succ 0$, if

$$\boldsymbol{p}_i^{\mathrm{T}}\boldsymbol{A}\boldsymbol{p}_j = 0 \qquad \text{for all } i \neq j.$$

Technique 4.4.2 ▸ Conjugate Gradient Method Algorithm

Let

$$\phi(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}$$

where $\boldsymbol{A} \succ 0$. Take $\boldsymbol{x}_0$ be the initial approximation for the global minimiser of ϕ and $\{\boldsymbol{p}_0, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_{n-1}\}$ be a set of directional vectors conjugate to $\boldsymbol{A}$. Set

$$\boldsymbol{r}_n = \boldsymbol{A}\boldsymbol{x}_n - \boldsymbol{b} = \nabla\phi(\boldsymbol{x}_n)$$

as the **residual**.

At the k-th iteration, consider

$$\begin{aligned} 0 &= \phi(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k)^{\mathrm{T}}\boldsymbol{p}_k \\ &= \left[\boldsymbol{A}(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k) - \boldsymbol{b}\right]^{\mathrm{T}}\boldsymbol{p}_k \\ &= \left(\boldsymbol{r}_k + \alpha\boldsymbol{A}\boldsymbol{p}_k\right)^{\mathrm{T}}\boldsymbol{p}_k \\ &= \boldsymbol{r}_k^{\mathrm{T}}\boldsymbol{p}_k + \alpha\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{p}_k. \end{aligned}$$

Therefore,

$$\alpha_k = \underset{\alpha>0}{\operatorname{argmin}}\, \phi(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) = -\frac{\boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{p}_k + \boldsymbol{p}_k}{\boldsymbol{p}_k^{\mathrm{T}} \boldsymbol{A}^{\mathrm{T}} \boldsymbol{p}_k}.$$

Take $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$.

Remark. Note that we can verify

$$\begin{aligned} \boldsymbol{r}_{k+1} &= \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b} \\ &= \boldsymbol{A}(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) - \boldsymbol{b} \\ &= \boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b} - \alpha_k \boldsymbol{A}\boldsymbol{p}_k \\ &= \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{p}_k \end{aligned}$$

To analyse the rate of convergence of conjugate gradient method, we consider the following theorem:

Theorem 4.4.3 ▸ Expanding Subspace Minimisation

Let $\boldsymbol{x}_k$ be the k-th approximation generated by the conjugate gradient method with $\{\boldsymbol{p}_0, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_{k-1}\}$ for the quadratic function

$$\phi(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x},$$

then $\boldsymbol{x}_k$ minimises ϕ over the set

$$\boldsymbol{x}_0 + \operatorname{span}\{\boldsymbol{p}_0, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_{k-1}\}.$$

Corollary 4.4.4 ▸ Convergence of Conjugate Gradient Method

For any $\boldsymbol{x}_0$ as the initial approximation for the minimiser of some quadratic function ϕ, the sequence $(\boldsymbol{x}_k)$ generated by the conjugate gradient method converges to the global minimiser of ϕ in at most n steps.

Proposition 4.4.5

At the $(k+1)$-th iteration of the conjugate gradient method,

$$\boldsymbol{p}_{k+1} = -\boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{p}_k,$$

where

$$\beta_{k+1} = \frac{\boldsymbol{r}_{k+1}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{p}_k}{\boldsymbol{p}_k^{\mathrm{T}} \boldsymbol{A} \boldsymbol{p}_k} = \frac{\boldsymbol{r}_{k+1}^{\mathrm{T}} \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{r}_k}, \qquad \boldsymbol{p}_0 = -\boldsymbol{r}_0.$$

Code Snippet 4.4.6 ▸ Conjugate Gradient Method Algorithm in Matlab

```
function x = conj_grad(x0, A, b, t)
    x = x0;
    r = A * x - b;
    p = -r;
    while norm(r) > t
        alpha = -(r' * p + p) / (p' * A' * p);
        x += alpha * p;
        r_new = r - alpha * A * p;
        beta = (r_new' * r_new) / (r' * r);
        p = -r_new + beta * p;
    end
end
```

5

Constrained Optimisation

5.1 Dual Optimisation Problems

Consider the non-linear programming problem

$$\begin{aligned} &\min_{\boldsymbol{x}\in\mathbf{R}^n} f(\boldsymbol{x}) \\ &\text{s.t. } g_i(\boldsymbol{x}) = 0 \quad \text{for } i = 1, 2, \cdots, m \\ &\qquad h_j(\boldsymbol{x}) \leq 0 \quad \text{for } j = 1, 2, \cdots, p. \end{aligned}$$

Previously, we have learnt how to optimise a function with no constraint. In non-rigorous language, if we were to convert this constrained problem to an unconstrained case, we would want to transform the objective function such that any points outside of the feasible set can never possibly be an optimal solution, i.e., choosing such a point will impose a penalty to our function value. To "get rid of" the constraints here, we may attempt to consider penalty functions

$$\min_{\boldsymbol{x}\in\mathbf{R}^n} \left[f(\boldsymbol{x}) + \sum_{i=1}^{m} P_i\big(g_i(\boldsymbol{x})\big) + \sum_{j=1}^{p} Q_j\big(h_j(\boldsymbol{x})\big) \right],$$

where

$$P_i\big(g_i(\boldsymbol{x})\big) = \begin{cases} \infty, & \text{if } g_i(\boldsymbol{x}) \neq 0 \\ 0, & \text{otherwise} \end{cases},$$

$$Q_j\big(h_j(\boldsymbol{x})\big) = \begin{cases} \infty, & \text{if } h_j(\boldsymbol{x}) > 0 \\ 0, & \text{otherwise} \end{cases}.$$

However, notice that by doing so our objective function has become **discontinuous**! To fix this, we may instead try linear functions like

$$P_i\big(g_i(\boldsymbol{x})\big) = |\lambda_i g_i(\boldsymbol{x})|, \qquad Q_j\big(h_j(\boldsymbol{x})\big) = \mu_j h_j(\boldsymbol{x}),$$

where $\lambda_i \in \mathbf{R}$ and $\mu_j \in \boldsymbol{R}_0^+$. One way to interpret this construct is that when our constraints are not satisfied, the P_i's and Q_j's will make our function value larger than $f(\boldsymbol{x})$. On the other hand, if our constraints are satisfied, the function value is even smaller than $f(\boldsymbol{x})$.

So an obvious pitfall of this transformation is that our optimal value will be altered. However, notice that if we take

$$P_i\big(g_i(\boldsymbol{x})\big) = \max_{\lambda_i \in \mathbf{R}} \lambda_i g_i(\boldsymbol{x}), \qquad Q_j\big(h_j(\boldsymbol{x})\big) = \max_{\mu_j \geq 0} \mu_j h_j(\boldsymbol{x}),$$

then we essentially get back to the first group of penalty functions!

Definition 5.1.1 ▶ Lagrangian Dual Problem

Let

$$\begin{aligned} &\min_{\boldsymbol{x} \in X} f(\boldsymbol{x}) \\ &\text{s.t. } g_i(\boldsymbol{x}) = 0 \quad \text{for } i = 1, 2, \cdots, m \\ &\qquad h_j(\boldsymbol{x}) \leq 0 \quad \text{for } j = 1, 2, \cdots, p \end{aligned}$$

be a constrained optimisation problem known as the **primal** problem. The function

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\mu}^{\mathrm{T}} \boldsymbol{h}(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x}) + \sum_{j=1}^{p} \mu_j h_j(\boldsymbol{x})$$

is known as the **Lagrangian** of the optimisation problem, where

$$\boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_m \end{bmatrix}, \qquad \boldsymbol{g} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_m \end{bmatrix}, \qquad \boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{bmatrix}, \qquad \boldsymbol{h} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_p \end{bmatrix}.$$

The **Lagrangian dual** problem is the optimisation problem

$$\max_{\lambda_i \in \mathbf{R}, \mu_j \geq 0} \left[\min_{\boldsymbol{x} \in X} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \right].$$

However, notice that along the "boundary" of the feasible set, the inequality constraints will transform into equality constraints. This motivates us to consider the feasible points along the boundary first.

In the following section, we shall define rigorously the meaning of a point being "on the boundary" of the feasible set.

5.2 Linear Independence Constraint Qualification

Definition 5.2.1 ▶ Active Constraint

An inequality constraint $h_j(\boldsymbol{x}) \leq 0$ is said to be **active** at $\boldsymbol{x}^* \in S$ if $h_j(\boldsymbol{x}^*) = 0$. Otherwise, it is said to be **inactive** or **slack** at $\boldsymbol{x}^*$.

The above definition essentially defines the notion of "being on the boundary" for any feasible point to a constrained NLP. Note that for any any $\boldsymbol{x}^*$ in the feasible set, any equality constraint $g_i(\boldsymbol{x}) = 0$ is always active.

Now, take any point $\boldsymbol{x}^*$ in the feasible set. If $\{h_1, h_2, \cdots, h_p\}$ are the set of inequality constraints of the NLP, then we can always find some subset $J \subseteq \{1, 2, \cdots, p\}$ such that h_j where $j \in J$ is active at $\boldsymbol{x}^*$. This J is known as an *index set*.

Definition 5.2.2 ▶ Regular Point

Let $\boldsymbol{x}^* \in S$ and let

$$J(\boldsymbol{x}^*) = \{j \in \{1, \cdots, p\} : h_j(\boldsymbol{x}^*) = 0\}$$

be the index set of active ineqaulity constraints at $\boldsymbol{x}^*$. If the set of gradient vectors

$$\{\nabla g_i(\boldsymbol{x}^*) : i = 1, 2, \cdots, m\} \cup \{\nabla h_j(\boldsymbol{x}^*) : j \in J(\boldsymbol{x}^*)\}$$

is linearly independent, then we say that $\boldsymbol{x}^*$ is a **regular point**. Alternatively, we say that the **Regularity Condition** or the **Linear Independence Constraint Qualification (LICQ)** holds at $\boldsymbol{x}^*$.

5.2.1 Special Case: Only Equality Constraints

If there is no inequality constraints, then $\boldsymbol{x}^*$ is a regular point if and only if the set

$$\{\nabla g_i(\boldsymbol{x}^*) : i = 1, 2, \cdots, m\}$$

is linearly independent. In practice, this means we can just check

$$\operatorname{rank}\left(\left[\nabla g_1(\boldsymbol{x}^*), \cdots, \nabla g_m(\boldsymbol{x}^*)\right]\right)$$

to determine the regularity. In particular, if there is only one equality constraint, i.e., $m = 1$, then we only need to check that $\nabla g_1(\boldsymbol{x}^*) \neq 0$. If there are two equality constraints, i.e., $m = 2$, then we check that there is no $\lambda \in \mathbb{R}$ such that $\nabla g_1(\boldsymbol{x}^*) = \lambda \nabla g_2(\boldsymbol{x}^*)$.

5.2.2 Special Case: Only Inequality Constraints

If there is no equality constraints, then $\boldsymbol{x}^*$ is a regular point if and only if

$$\left\{\nabla h_j(\boldsymbol{x}^*) : \ j \in J(\boldsymbol{x}^*)\right\}$$

is linearly independent. In particular, notice that if $\boldsymbol{x}^*$ is an interior point of the feasible set, then none of the inequality constraints is active, which means the set $\left\{\nabla h_j(\boldsymbol{x}^*) : \ j \in J(\boldsymbol{x}^*)\right\}$ is empty. However, recall that $\varnothing$ is linearly independent, so any interior point $\boldsymbol{x}^*$ is always a regular point.

5.3 Karush-Kuhn-Tucker (KKT) Points

We are now ready to generalise the Lagrange multiplier method into the KKT conditions.

Definition 5.3.1 ▸ KKT Points

Let $\boldsymbol{x}^*$ be a regular point. $\boldsymbol{x}^*$ is a **KKT point** if:

- there are scalars $\lambda_1, \cdots, \lambda_m$ and $\mu_1, \cdots, \mu_p$, called the **KKT multipliers**, such that
$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\boldsymbol{x}^*) + \sum_{j=1}^{p} \mu_j \nabla h_j(\boldsymbol{x}^*) = \boldsymbol{0};$$
- $g_i(\boldsymbol{x}^*) = 0$ for $i = 1, 2, \cdots, m$ and $h_i(\boldsymbol{x}^*) \leq 0$ for $i = 1, 2, \cdots, p$;
- $\mu_i \geq 0$ for $i = 1, 2, \cdots, p$;
- $\mu_i = 0$ for $i \notin J(\boldsymbol{x}^*)$,

where $J(\boldsymbol{x}^*)$ is the index set of active inequality constraints at $\boldsymbol{x}^*$.

5.3.1 Special Case: Only One Inequality Constraint

If there is only one inequality constraint h_1 and no equality constraint, then $\boldsymbol{x}^*$ is a KKT Point if

$$\nabla f(\boldsymbol{x}^*) + \mu_1 \nabla h_1(\boldsymbol{x}^*) = 0$$

for some $\mu_1 \geq 0$. This means that $-\nabla f(\boldsymbol{x}^*)$ is in exactly the same direction as $\nabla h_1(\boldsymbol{x}^*)$.

5.3.2 Special Case: Only Two Inequality Constraints

If there is only two inequality constraints h_1 and h_2 and no equality constraint, then $\boldsymbol{x}^*$ is a KKT Point if

$$\nabla f(\boldsymbol{x}^*) + \mu_1 \nabla h_1(\boldsymbol{x}^*) + \mu_2 \nabla h_2(\boldsymbol{x}^*) = 0$$

for some $\mu_1 \geq 0$. This means that $-\nabla f(\boldsymbol{x}^*) \in \text{span}\{\nabla h_1(\boldsymbol{x}^*), \nabla h_2(\boldsymbol{x}^*)\}$. Note that this space is a cone spanned by the outward normals to the curves $h_1(\boldsymbol{x}) = 0$ and $h_2(\boldsymbol{x}) = 0$.

5.3.3 Special Case: Interior Points

If the local minimiser $\boldsymbol{x}^*$ is an interior point of the feasible set and there is no equality constraint, then $\boldsymbol{x}^*$ is a KKT Point if

$$\nabla f(\boldsymbol{x}^*) = 0.$$

Therefore, similar to an unconstrained NLP, $\boldsymbol{x}^*$ is a stationary point.

5.4 KKT Conditions

Before we state the KKT conditions formally, we first consider the following preliminary definition:

Definition 5.4.1 ▸ Critical Cone

Let $\boldsymbol{x}^*$ be a KKT point such that

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\boldsymbol{x}^*) + \sum_{j=1}^{p} \mu_j \nabla h_j(\boldsymbol{x}^*) = 0,$$

then the set

$$C(\boldsymbol{x}^*, \lambda, \mu) := \left\{ \boldsymbol{y} \in \mathbf{R}^n : \begin{array}{ll} \nabla g_i(\boldsymbol{x}^*)^{\mathrm{T}} \boldsymbol{y} = 0 & \text{for } i = 1, 2, \cdots, m \\ \nabla h_j(\boldsymbol{x}^*)^{\mathrm{T}} \boldsymbol{y} = 0 & \text{for } j \in J(\boldsymbol{x}^*), \mu_j > 0 \\ \nabla h_j(\boldsymbol{x}^*)^{\mathrm{T}} \boldsymbol{y} \leq 0 & \text{for } j \in J(\boldsymbol{x}^*), \mu_j = 0 \end{array} \right\}$$

is known as the **critical cone** at $\boldsymbol{x}^*$.

Observe that

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\boldsymbol{x}^*) + \sum_{j=1}^{p} \mu_j \nabla h_j(\boldsymbol{x}^*)$$

is essentially $\nabla \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu})$. Therefore, we can naturally find the Hessian of $\mathcal{L}$ as

$$H_{\mathcal{L}}(\boldsymbol{x}^*) = H_f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i H_{g_i}(\boldsymbol{x}^*) + \sum_{j=1}^{p} \mu_j H_{h_j}(\boldsymbol{x}^*).$$

Theorem 5.4.2 ▶ Kurash-Kuhn-Tucker (KKT) Conditions

Consider the optimisation problem

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & g_i(\boldsymbol{x}) = 0 \quad \text{for } i = 1, 2, \cdots, m \\ & h_j(\boldsymbol{x}) \leq 0 \quad \text{for } j = 1, 2, \cdots, p. \end{aligned}$$

Suppose that f and each of the g_i's and h_j's have continuous first order partial derivatives. Let $\boldsymbol{x}^*$ be a regular point.

*The **Kurash-Kuhn-Tucker First Order Necessary Condition** states that if $\boldsymbol{x}^*$ is a local minimiser, then $\boldsymbol{x}^*$ is a KKT point.*

*The **Kurash-Kuhn-Tucker Second Order Necessary Condition** states that if f and each of the g_i's and h_j's have continuous second order partial derivatives, then $\boldsymbol{x}^*$ is a KKT point and*

$$\boldsymbol{y}^{\mathrm{T}} H_{\mathcal{L}}(\boldsymbol{x}^*)\boldsymbol{y} \geq 0$$

for all $\boldsymbol{y} \in C(\boldsymbol{x}^*, \lambda, \mu)$, where

$$\mathcal{L}(\boldsymbol{x}^*) = \nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\boldsymbol{x}^*) + \sum_{j=1}^{p} \mu_j \nabla h_j(\boldsymbol{x}^*).$$

We first justify the first order condition. Notice that the condition $\mu_j \geq 0$ can be argued in the following manner:

If $\boldsymbol{x}^*$ is an interior point, then the problem is reduced to an optimisation problem with only equality constraints. By the Lagragian multiplier method, $\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x}^*) = 0$. Since $\boldsymbol{x}^*$ is a regular point, the set

$$\left\{\nabla h_j(\boldsymbol{x}^*) : \ j = 1, 2, \cdots, p\right\}$$

is linearly independent. Therefore, all of μ_j's must be identically 0.

If $\boldsymbol{x}^*$ is on the boundary of the feasible set and there is some $\mu_j < 0$, this means that the vector $\nabla f(\boldsymbol{x}^*)$ contains some component in the outwards direction of $h_j(\boldsymbol{x}^*) = 0$. However, this means f decreases along the inward direction from the boundary of the feasible set, so $\boldsymbol{x}^*$ is not a local minimiser, which is a contradiction.

Note that the condition $\mu_i = 0$ for $i \in J(\boldsymbol{x}^*)$ is equivalent to $\mu_j h_j = 0$ for aall $j \in \mathbb{N}^+$. The latter is known as the **Complementary Slackness Condition**. The reasoning is as follows:

If h_j is active at $\boldsymbol{x}^*$, then $h_j(\boldsymbol{x}^*) = 0$. Otherwise, h_j is slack and so we have $\mu_j = 0$. Either way, $\mu_j h_j(\boldsymbol{x}^*) = 0$.

For the second order necessary condition, we may wish to argue why it suffices to consider the critical cone only.

Notice that for all $\boldsymbol{z} \notin C(\boldsymbol{x}^*, \lambda, \mu)$, either of the following is satisfied:

Case I: $\nabla g_i(\boldsymbol{x}^*)^{\mathrm{T}} \boldsymbol{z} \neq 0$ for some i. In this case, the direction of $\boldsymbol{z}$ is not along the constraint curve $g_i(\boldsymbol{x}) = 0$, so it is irrelevant to the NLP.

Case II: $\mu_j > 0$ for some $j \in J(\boldsymbol{x}^*)$ but $\nabla h_j(\boldsymbol{x}^*)^{\mathrm{T}} \boldsymbol{z} \neq 0$. In this case, the direction of $\boldsymbol{z}$ is either towards the interior of the feasible set, along which the value of f increases, or outwards from the boundary of the feasible set. Either way, there cannot be an optimal solution along the direction of $\boldsymbol{z}$.

Case III: $\mu_j = 0$ for some $j \in J(\boldsymbol{x}^*)$ but $\nabla h_j(\boldsymbol{x}^*)^{\mathrm{T}} \boldsymbol{z} > 0$. In this case, the direction of $\boldsymbol{z}$ is outwards from the feasible set and so it is irrelevant to consider any point along the direction of $\boldsymbol{z}$.

5.5 Constrained Convex Optimisation Problems

Definition 5.5.1 ▶ Constrained Convex Non-linear Programming Problem

A **constrained convex non-linear programming problem** is defined as

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & g_i(\boldsymbol{x}) = \boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x} - \boldsymbol{b} = 0 \quad \text{for } i = 1, 2, \cdots, m \\ & h_j(\boldsymbol{x}) \leq 0 \quad \text{for } j = 1, 2, \cdots, p, \end{aligned}$$

where f and the h_j's are convex functions and the g_i's are linear functions.

Theorem 5.5.2 ▶ Optimum of Constrained Convex NLPs

Let f be an objective function with equality constraints $g_i(\boldsymbol{x}) = 0$ for $i = 1, 2, \cdots, m$ and ineqaulity constraints $h_j(\boldsymbol{x}) \leq 0$ for $j = 1, 2, \cdots, p$. If f and the h_j's are differentiable convex functions and $g_i(\boldsymbol{x}) = \boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x} - \boldsymbol{b}$ is linear for $i = 1, 2, \cdots, m$. If $\boldsymbol{x}^*$ is a feasible KKT point, then it is a global minimiser for f.

Definition 5.5.3 ▶ Slater's Condition

For a convex programming problem with at least one inequality constraint and feasible set S, the **Slater's Condition** holds if there is some $\hat{\boldsymbol{x}} \in S$ such that $h_j(\hat{\boldsymbol{x}}) < 0$ for all $j = 1, 2, \cdots, p$.

Remark. In other words, Slater's condition verifies that there is at least one point which is **strictly feasible**, i.e., the interior of the feasible set is **non-empty**.

6

Lagragian Dual Problems

We would like to show that

$$\min_{x\in X} f(x) \quad \text{s.t. } g_1(x) \equiv \min_{x\in X}\left\{\max_{\lambda_1\in \mathbf{R}}[f(x+\lambda_1 g_1(x))]\right\}.$$

Proposition 6.0.1

$$\max_{\lambda_1\in \mathbf{R}}[f(x+\lambda_1 g_1(x))] = \begin{cases} f(x), & \text{if } g_1(x)=0 \\ \infty, & \text{otherwise} \end{cases},$$

$$\max_{\mu_1\in \mathbf{R}}[f(x+\mu_1 h_1(x))] = \begin{cases} f(x), & \text{if } h_1(x)\le 0 \\ \infty, & \text{otherwise} \end{cases}.$$

$$\min_{x\in X}\left\{\max_{\lambda\in \mathbf{R}^m,\mu\in \mathbf{R}^p_+}\left[f(x)+\lambda^{\mathrm{T}} g(x)+\mu^{\mathrm{T}} h(x)\right]\right\}$$

Lagrangian function:

$$\mathcal{L}(x,\lambda,\mu) = f(x)+\lambda^{\mathrm{T}} g(x)+\mu^{\mathrm{T}} h(x).$$

Lagrangian dual function:

$$\theta(\lambda,\mu) = \inf_{x\in X}\mathcal{L}(x,\lambda,\mu).$$