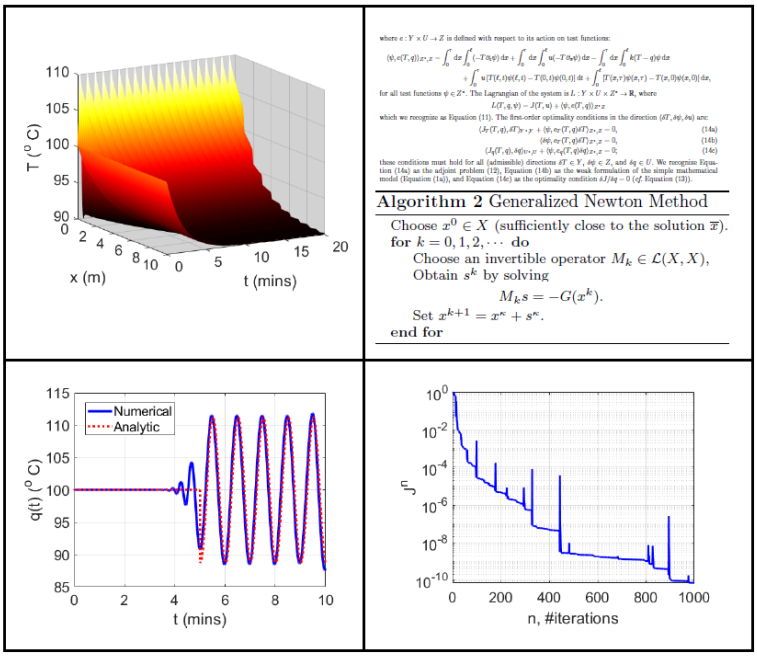
University College Dublin
An Coláiste Ollscoile, Baile Átha Cliath

School of Mathematics and Statistics
Scoil na Matamaitice agus Staitisticí
Optimization Algorithms

For use in Optimization and Machine Learning (ACM 40990)

and Optimization Algorithms (ACM41030)



Dr Lennon Ó Náraigh

Lecture Notes in Optimization Algorithms
January 2026

Optimization in Machine Learning (ACM 40990)

- Subject: Applied and Computational Mathematics
- School: Mathematics and Statistics
- Module coordinator: Dr Lennon Ó Náraigh
- Credits: 5
- Level: 4
- Semester: Spring

This module introduces students to the mathematical techniques that form the cornerstone of Machine Learning. Students will first of all study in depth the key concepts in continuous optimization (unconstrained, constrained, and global), before going on to apply these concepts to common algorithms in Machine Learning.

Topics covered: Steepest-Descent and Newton-type methods, including analysis of convergence, Trust-region methods, including the construction of solutions of the constrained sub-problem. Numerical implementations of standard optimization methods. Necessary first-order optimality conditions. Introduction to Global Optimization, to include a discussion on Simulated Annealing. Application of optimization techniques through worked examples in Python. Examples may include: Linear Regression, Matrix Completion and Compressed Sensing, Support Vector Machines, and Neural Networks.

What will I learn?

On completion of this module students should be able to

1. Formulate standard optimization techniques in continuous optimization, understand the convergence criteria, and implement these methods from scratch;
2. Understand the first-order necessary conditions for optimality in constrained optimization, be able to solve simple problems by hand
3. Understand the need for global optimization, implement a simulated-annealing algorithm
4. Using Python programming, apply optimization techniques to problems in Machine Learning

Optimization Algorithms (ACM41030)

- Subject: Applied and Computational Mathematics
- School: Mathematics and Statistics
- Module coordinator: Dr Lennon Ó Náraigh;
- Credits: 5
- Level: 4
- Semester: Spring

This module introduces students to the theory of optimization, a key tool in modern Applied Mathematics, Operations Research, and Machine Learning. Students will study in depth the key concepts in continuous optimization - both unconstrained, constrained, and global.

Topics covered: Steepest-Descent and Newton-type methods, including analysis of convergence, Trust-region methods, including the construction of solutions of the constrained sub-problem, **Non-Linear Least Squares, including the Levenberg-Marquardt method**. Numerical implementations of standard optimization methods. **Constrained optimization with equality and inequality constraints, examples motivating the introduction of the Lagrange Multiplier Technique. Necessary first-order optimality conditions, including a derivation of the Karush-Kuhn-Tucker conditions. Farkas's lemma and the Separating Hyperplane Theorem.** Introduction to Global Optimization, to include a discussion on Simulated Annealing.

On completion of this module students should be able to

1. Formulate standard optimization techniques in continuous optimization, understand the convergence criteria, and implement these methods from scratch;
2. Implement the same methods using standard software packages, understand when these methods will work well and when they won't;
3. Understand the first-order necessary conditions for optimality in constrained optimization, be able to solve simple problems by hand
4. Sketch the proof of the Karush-Kuhn-Tucker conditions
5. Understand the need for global optimization, implement a simulated-annealing algorithm

Editions

First edition: January 2023

Second edition: January 2024

This edition: January 2025

Acknowledgement of Source Materials

The lecture notes on local techniques for optimization (unconstrained and constrained) are based almost entirely on the book 'Numerical Optimization', by Jorge Nocedal and Stephen J. Wright [1].

The lecture notes on Global Optimization with Simulated Annealing are based on my own research experiences, a journal article by Ingber [2], as well as the following books:

- 'Statistical Physics' by F. Mandl [3];
- 'Handbook of Metaheuristics' M. Gendreau and J. Y. Potvin [4].

The lecture notes on the applications in Machine Learning are based on materials developed over the years by Barry Wardell and Jake Williams (UCD), and Marco Viola (DCU). I have also used the book 'Mathematics for Machine Learning' by Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong [5], as well as the key paper 'Deep Learning: An Introduction for Applied Mathematicians', by Catherine F. Higham and Desmond J. Higham [6].

Contents

Chapter 1

Introduction to Optimization

Overview

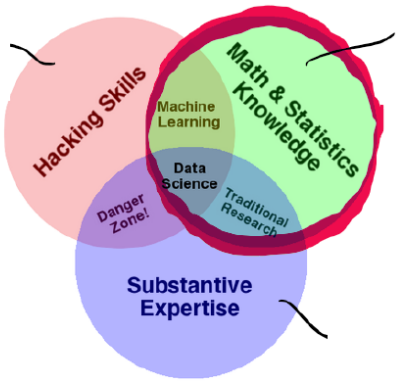
We introduce the key concepts in Optimization. We also explain the rationale for studying the topic, in the context of Data Science.

1.1 Why optimization?

Optimization is at the heart of many of the techniques in Machine Learning, this is a solid rationale for studying optimization in the context of Data Science. In this module, we are going to look at the theoretical underpinnings for the main methods in (continuous) Optimization. Why should we look at the theory? This is a fair question, when languages such as Matlab and Python contain many optimization algorithms that can be coded up in a couple of minutes. Yet the answer to the question is already contained in this statement – these codes are 'black boxes', without understanding the rudiments of what is in these black boxes, we run the risk of GIGO – this idea is also what is going on in the background in Figure 1.1. So for these reasons, it is important to understand the theory of what is in these black boxes, in these optimization algorithms.

The module is broken into two parts:

- First 7 weeks – we look at the mathematical theory of continuous optimization, including both local optimization and global optimization.
- Remaining 5 weeks:
  - We look at the implementation of the various optimization techniques in Machine-Learning algorithms (ACM40990)
  - We look at the theory of constrained optimization (ACM41030)



http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram

Figure 1.1: Without mathematical knowledge of what is going on in optimization algorithms, blind application of those algorithms can lead us into the 'danger zone'.

1.1.1 Wider Context

Optimization is used more widely in Industry but also is a key concept in understanding natural phenomena. In industry, Optimization is used, e.g.

- Investment – portfolio managers try to minimize the risk of a portfolio while seeking a high rate of return;
- Manufacturing – managers try to maximize output while keeping input costs down.

In nature, systems tend to a state of minimum energy (equilibrium). For instance, soap films tend to a configuration of minimum surface area, thereby minimizing energy. Rays of light follow paths that minimize the travel time. In these cases, optimization can be used to convert these general principles into sets of equations and hence, to make predictions.

1.2 Terminology

The following is absolutely basic terminology, which will be used throughout the module:

- $\boldsymbol{x} \in \mathbb{R}^n$ is the vector of **variables**, also referred to as the unknowns or parameters.
- f is the **objective function** (or cost function or penalty function). The objective function is a scalar-valued function of $\boldsymbol{x}$; we seek a minimum or a maximum value of f.
- c_i are **constraint functions**, these are scalar functions of $\boldsymbol{x}$ that define certain equations (or inequalities); the unknown vector $\boldsymbol{x}$ must satisfy these equations or inequalities.

Using ths notation, the generic optimization problem to be studied in this module is:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \text{ subject to } \begin{cases} c_i(\boldsymbol{x}) = 0, & i \in \mathcal{E}, \\ c_i(\boldsymbol{x}) \geq 0, & i \in \mathcal{I}. \end{cases} \tag{1.1}$$

Here, $\mathcal{E}$ and $\mathcal{I}$ are sets of indices for equality and inequality constraints, respectively.

Example: Consider the problem:

$$\min(x_1 - 2)^2 + (x_2 - 1)^2 \text{ subject to } \begin{cases} x_1^2 - x_2 \leq 0, \\ x_1 + x_2 \leq 2. \end{cases} \tag{1.2}$$

We identify $\boldsymbol{x} \in \mathbb{R}^2$, $\boldsymbol{x} = (x_1, x_2)^T$. The objective function is:

$$f(\boldsymbol{x}) = (x_1 - 2)^2 + (x_2 - 1)^2.$$

The constraint functions can be written as:

$$c(\boldsymbol{x}) = \begin{pmatrix} c_1(\boldsymbol{x}) \\ c_2(\boldsymbol{x}) \end{pmatrix} = \begin{pmatrix} -x_1^2 + x_2 \\ -x_1 - x_2 + 2 \end{pmatrix},$$

hence $\mathcal{E} = \emptyset$, and $\mathcal{I} = \{1, 2\}$.

The solution can be determined graphically from Figure 1.2. Clearly, the solution is:

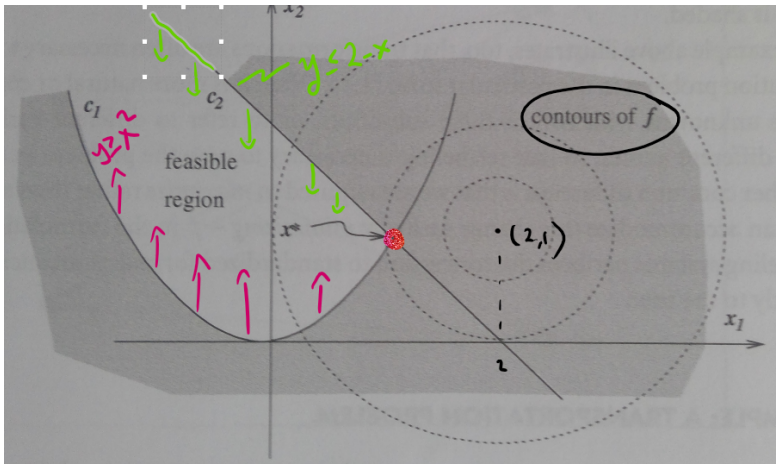


Figure 1.2: Graphical interpretation of the Optimization Problem (OP) (1.2)

- As close as possible to the zero level set of f, which is the centre of the family of circles shown in the figure (hence the point $(2, 1)$);

- Inside the **feasible region**

Therefore, the required minimum is at the intersection of the curves c_1 and c_2. We take $x_1 := x$ and $x_2 := y$ for easy notation, and we solve for $c_1(x, y) = c_2(x, y)$:

$$y = x^2, \qquad y = 2 - x \implies x^2 = 2 - x,$$

hence $x^2 + x - 2{=}0$. The quadratic formula (positive branch) then yields $x = 1$, and $y = 2 - x$, hence $y = 1$, hence finally:

$$\boldsymbol{x}_* = (1, 1)^T .$$

This solution is checked numerically using Matlab (see listings below):

```
function x_star=op1()

x0=[1;1];

fval=@myfun;
nonlcon=@mycon;

A=[1,1];
b=2;

x_star = fmincon(fval,x0,A,b,[],[],[],[],nonlcon);

    function y=myfun(x)
        y=(x(1)-2)^2+(x(2)-1)^2;
    end

    function [c,ceq] = mycon(x)
        c = x(1)^2 -x(2);
        ceq = [];
    end

end
```

Definition 1.1 *The* **feasible region** *is the set of all points satisfying the constraints.*

Remark: For inequality constraints, the feasible region is the area between the constraint boundaries.

Notation: We denote the solution of the optimization problem by $\boldsymbol{x}_*$.

Example: A chemical company has the following set-up:

- 2 factories F_1 and F_2;
- 12 retail outlets $R_1, R_2, \cdots, R_{12}$.

Furthermore,

- Each factory F_i can produce a_i tons of product per week (**the capacity** of plant F_i).
- Each retail outlet R_j has a known weekly demand b_j tons of product.

Let x_{ij} denote the number of tons shipped from factory F_i to retail outlet R_j per week. Also, let c_{ij} be the cost of shipping one tone of product. Write down the optimization problem for minimizing the shipping cost.

Solution: The cost function is the cost of shipping x_{ij} tons at cost c_{ij} from factory i to retail outlet j, summed over all i and j:

$$f = \sum_{i=1}^{2} \sum_{j=1}^{12} x_{ij} c_{ij}. \tag{1.3a}$$

But there are constraints:

- Factory i:

$$\sum_{j=1}^{12} x_{ij} \leq a_i, \qquad \text{for all } i = 1, 2; \tag{1.3b}$$

- Retail outlet j – meeting the demand:

$$\sum_{i=1}^{2} x_{ij} = b_j, \tag{1.3c}$$

  or exceeding the demand:

$$\sum_{i=1}^{2} x_{ij} \geq b_j, \tag{1.3d}$$

- Positivity:

$$x_{ij} \geq 0, \qquad \text{for all } i = 1, 2, \text{ and for all } j = 1, \cdots, 12. \tag{1.3e}$$

Notice that we can re-write the array x_{ij} as a $2 \times 12 = 24$ vector $\boldsymbol{x}$. Thus, the cost function and constraint functions are linear functions in $\boldsymbol{x} \in \mathbb{R}^{24}$, and hence, OP (23.19) is an example of **linear programming**.

1.2.1 Continuous versus discrete optimization

Often, instead of seeking a solution of $\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$, we are interested in optimizing over $\boldsymbol{x} \in \mathbb{N}_0^n$ or even over $\boldsymbol{x} \in \{0, 1\}^n$. Here, we are using the notation $\mathbb{N}_0 = \{0, 1, 2, \cdots\}$. Examples:

- $\boldsymbol{x} \in \mathbb{N}_0^n$: x_i is the number of power plants of type i that the grid operator should construct over the next five years, $i \in \{1, 2, \cdots, n\}$;

- $\boldsymbol{x} \in \{0,1\}^n$: x_i is a binary variable, and indicates whether or not a factory should be located in city i.

These are both examples of **linear programming**, which is a type of **discrete optimization**. We mention discrete optimization here for completeness, however, the focus of this course is on **continuous optimization**, where we solve OP (1.1).

1.2.2 Global versus local optimization

We will focus a considerable effort on understanding so-called steepest-descent algorithms (SD) for solving OP (1.1). SD algorithms are good for finding a single solution of the OP – hence, a **local minimum**. In practice, it is difficult (e.g. in a high-dimensional OP) to know if the computed minimum is a global minimum. For so-called **convex problems**, there is a unique minimum, hence, any computed minimum is the unique global minimum. For this purpose, we have to define convexity. As well as looking at SD-type algorithms, in the later part of the course, we will also look at metaheuristic algorithms for computing the global optimum.

1.3 Convex sets

Convexity is the fundamental concept in Optimization.

Definition 1.2 *Let $S \subset \mathbb{R}^n$. S is called a convex set if a straight line segment joining any two points in S is itself contained entirely in S.*

In symbols, S is convex if, for each $\boldsymbol{x}$ and $\boldsymbol{y}$ in S, the line segment

$$\boldsymbol{x}t + (1-t)\boldsymbol{y}, \qquad t \in [0,1],$$

is contained entirely in S.

See Figure 1.3.

1.3.1 The Unit Ball

Theorem 1.1 *The unit ball is a convex set:*

$$B = \{\boldsymbol{x} \in \mathbb{R}^n | \|\boldsymbol{x}\|_2^2 \leq 1\}.$$

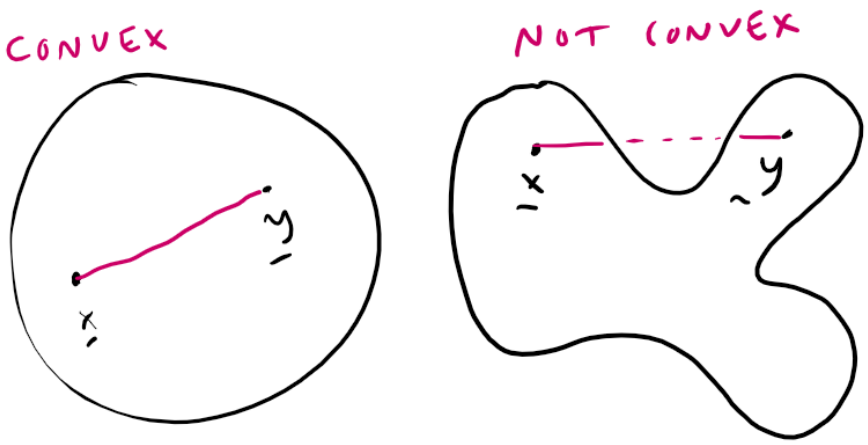


Figure 1.3: Different convex and non-convex sets

Proof: To prove this statement, we require prior knowledge of linear algebra and dot products. We take the dot product to be the usual dot product on $\mathbb{R}^n$, we will sometimes use 'angle bracket' notation for this:

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^{n} x_i y_i.$$

Then, the L^2 norm of a vector is denoted by $\|\boldsymbol{x}\|_2^2$ and is given by:

$$\|\boldsymbol{x}\|_2^2 = \sum_{i=1}^{n} x_i^2 = \langle \boldsymbol{x}, \boldsymbol{x} \rangle.$$

Therefore, we can verify that the unit ball is a convex set as follows. For, let $\boldsymbol{x}$ and $\boldsymbol{y}$ both be in B. Then consider the path

$$\boldsymbol{x}(t) = \boldsymbol{x}t + \boldsymbol{y}(1-t), \qquad t \in [0,1].$$

We compute:

$$\langle \boldsymbol{x}(t), \boldsymbol{x}(t) \rangle = \langle \boldsymbol{x}, \boldsymbol{x} \rangle t^2 + 2t(1-t)\langle \boldsymbol{x}, \boldsymbol{y} \rangle + \langle \boldsymbol{y}, \boldsymbol{y} \rangle (1-t)^2.$$

Here, we have used the distributivity properties of dot product. We now use the **Cauchy–Schwarz** inequality, along with $\langle \boldsymbol{x}, \boldsymbol{x} \rangle \leq 1$ and $\langle \boldsymbol{y}, \boldsymbol{y} \rangle \leq 1$:

$$\begin{aligned}
\langle \boldsymbol{x}(t), \boldsymbol{x}(t) \rangle &\leq (1)t^2 + 2t(1-t)\langle \boldsymbol{x}, \boldsymbol{y} \rangle + (1-t)^2(1), \\
&\leq t^2 + 2t(1-t)\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2 + (1-t)^2, \\
&\leq t^2 + 2t(1-t) + (1-t)^2, \\
&\leq t^2 + 2t - 2t^2 + 1 - 2t + t^2, \\
&= 1.
\end{aligned}$$

Hence, $\langle \boldsymbol{x}(t), \boldsymbol{x}(t) \rangle \leq 1$, hence $\boldsymbol{x}(t) \in B$. ∎

1.3.2 Polyhedra

Theorem 1.2 *Any polyhedron, i.e. any set defined by linear constraints, e.g.*

$$S = \{\boldsymbol{x} \in \mathbb{R}^n | A\boldsymbol{x} = b, C\boldsymbol{x} \leq d\},$$

where A and C are matrices of appropriate dimensions, and b and d are vectors, is a convex set.

The proof is left as an exercise.

1.4 Convex Functions

Definition 1.3 *Let $f : (S \subset \mathbb{R}^n) \to \mathbb{R}$ be a function. f is a* **convex function** *if:*

1. *S is a convex set;*
2. *The following relation holds:*

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

for all $t \in [0, 1]$ and for all $\boldsymbol{x}$ and $\boldsymbol{y}$ in S.

1.4.1 Linear Functions

Theorem 1.3 *Let $S = \mathbb{R}^n$. Then the linear function*

$$f(\boldsymbol{x}) = \langle \boldsymbol{b}, \boldsymbol{x} \rangle + \alpha,$$

is a convex function, here $\boldsymbol{b}$ is a constant vector and α is a constant scalar.

1.4.2 Quadratic Functions

A key example of a convex function in Optimization is the quadratic function. Again, let $S = \mathbb{R}^n$, and consider:

$$f(\boldsymbol{x}) = \langle \boldsymbol{x}, H\boldsymbol{x} \rangle, \tag{1.4}$$

where $H \in \mathbb{R}^n$ is a symmetric positive semi-definite matrix.

Theorem 1.4 *$f(\boldsymbol{x})$ in Equation (1.4) is a convex function.*

Proof: Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be vectors, and consider the line segment $\boldsymbol{x}t + \boldsymbol{y}(1-t)$, where $\in [0,1]$. We compute:

$$\begin{aligned} f(t\boldsymbol{x}+(1-t)\boldsymbol{y}) &= \langle t\boldsymbol{x}+(1-t)\boldsymbol{y}, tH\boldsymbol{x}+(1-t)H\boldsymbol{y}\rangle, \\ &= t^2\langle \boldsymbol{x}, H\boldsymbol{x}\rangle + 2t(1-t)\langle \boldsymbol{x}, H\boldsymbol{y}\rangle + (1-t)^2\langle \boldsymbol{y}, H\boldsymbol{y}\rangle, \qquad (*) \end{aligned}$$

Consider also:

$$tf(\boldsymbol{x}) + (1-t)f(\boldsymbol{y}) = t\langle \boldsymbol{x}, H\boldsymbol{x}\rangle + (1-t)\langle \boldsymbol{y}, H\boldsymbol{y}\rangle. \qquad (**).$$

Now take $(*) - (**)$:

$$\begin{aligned} (*)-(**) &= -t(1-t)\langle \boldsymbol{x}, H\boldsymbol{x}\rangle + 2t(1-t)\langle \boldsymbol{x}, H\boldsymbol{y}\rangle - t(1-t)\langle \boldsymbol{y}, H\boldsymbol{y}\rangle, \\ &= t(1-t)\left[-\langle \boldsymbol{x}, H\boldsymbol{x}\rangle + 2\langle \boldsymbol{x}, H\boldsymbol{y}\rangle - \langle \boldsymbol{y}, H\boldsymbol{y}\rangle\right], \\ &= -t(1-t)\left[\langle \boldsymbol{x}, H\boldsymbol{x}\rangle - 2\langle \boldsymbol{x}, H\boldsymbol{y}\rangle + \langle \boldsymbol{y}, H\boldsymbol{y}\rangle\right], \\ &= -\underbrace{t(1-t)}_{\text{pos}}\underbrace{\langle \boldsymbol{x}-\boldsymbol{y}, H(\boldsymbol{x}-\boldsymbol{y})\rangle}_{\text{pos}}, \\ &= \leq 0. \end{aligned}$$

Hence, $(*) \leq (**)$, hence

$$f(t\boldsymbol{x}+(1-t)\boldsymbol{y}) \leq tf(\boldsymbol{x}) + (1-t)f(\boldsymbol{y}). \quad \blacksquare$$

1.5 Optimization Algorithms

In this course, we are focused on solving the OP (1.1), which is a continuous problem. We will be solving problems in high-dimensional spaces, such that the graphical method employed for OP (1.2) is not useful. In these cases, a numerical optimization method is required. Such numerical optimization methods tend to be **iterative**, that is, they involve an initial guess, an improved guess, which becomes the new initial guess, which is then used to generate a new improved guess and so on. The improved guess is computed using properties of the cost function and the constraints – generally the derivative of the cost function (and sometimes the second derivative). If the distance between successive guesses goes to zero (in the sense of the L^2 norm) as the number of iterations goes to infinity, the method is said to **converge**. Besides convergence, we will look for the following 'good' properties in an optimization method:

- Robustness – the method should be reliable for a wide class of problem and for a wide class of initial guesses.
- Efficiency – the method should not require 'excessive' computer time or storage.
- Accuracy – the method should be able to identify a solution with precision, without being too sensitive to errors in the data or to arithmetic rounding errors.

Chapter 2

Fundamentals of Unconstrained Optimization

Overview

We look at the solution of optimization problems in the cases of continuously differentiable cost functions. We formulate necessary and sufficient conditions at optimality in terms of the derivatives of the cost function.

2.1 Definitions

We start with some definitions. We have in find the following OP:

$$\min_{\boldsymbol{x}\in S} f(\boldsymbol{x}), \qquad S \subset \mathbb{R}^n.$$

The solution $\boldsymbol{x}_*$ is written as:

$$\boldsymbol{x}_* = \arg\min_{\boldsymbol{x}\in S} f(\boldsymbol{x}), \qquad S \subset \mathbb{R}^n.$$

and we characterize the possible solutions of the OP as follows.

Definition 2.1 (Minimizers)

- $\boldsymbol{x}_*$ is a **Global Minimzer** if

$$f(\boldsymbol{x}_*) \leq f(\boldsymbol{y}), \qquad \text{for all } \boldsymbol{y} \in S.$$

- $\boldsymbol{x}_*$ is a **Local Minimzer** if there exists a **neighbourhood** $\mathcal{N} \subset S$ such that

$$f(\boldsymbol{x}_*) \leq f(\boldsymbol{y}), \qquad \text{for all } \boldsymbol{y} \in \mathcal{N}.$$

  Here, by a neighbourhood, we mean a non-empty, open set.

- $\boldsymbol{x}_*$ is a **Strict Local Minimzer** if there exists a neighbourhood $\mathcal{N} \subset S$ such that

$$f(\boldsymbol{x}_*) < f(\boldsymbol{y}), \qquad \text{for all } \boldsymbol{y} \in \mathcal{N}, \boldsymbol{y} \neq \boldsymbol{x}_*.$$

Example: The constant function $f(\boldsymbol{x}) = 1$, $\boldsymbol{x} \in \mathbb{R}^n$. Hence, every $\boldsymbol{x} \in \mathbb{R}^n$ is a local minimizer. But consider $f(x) = (x-2)^4$, $x \in \mathbb{R}$. Here, $f(x)$ has a strict local minimizer.

- $\boldsymbol{x}_*$ is an **Isolated Local Minimizer** if there exists a neighbourhood $\mathcal{N}$ of $\boldsymbol{x}_*$ such that x_* is the only local minimizer in $\mathcal{N}$.

Example: The function

$$f(x) = \begin{cases} x^4 \cos(1/x) + 2x^4, & x \neq 0, \\ 0, & x = 0, \end{cases}$$

is twice continuously differentiable and has a strict local minimizer at $x_* = 0$. However, there are strict local minimizers at many nearby points x_j, and we can label these points so that $x_j \to 0$ as $j \to \infty$. This is a (pathological) example of a function with a strict local minimum that is not isolated.

2.2 Necessary Conditions

We derive the necessary conditions for optimality in the case of $f : (S \subset \mathbb{R}) \to \mathbb{R}$. We then generalize by analogy to $f : (S \subset \mathbb{R}^n) \to \mathbb{R}$, with $n > 1$. The starting-point is Taylor's Remainder Theorem:

Theorem 2.1 *Let $f : \mathbb{R} \to \mathbb{R}$ be continuous function with a continuous first derivative. Assume $h > 0$. Then, there exists an $\eta \in (0, h)$ such that:*

$$f(x+h) = f(x) + f'(x+\eta)h, \qquad \eta \in (0, h).$$

Furthermore, if f is twice differentiable, with a continuous second derivative, then for a given $h > 0$, there exists an $\eta \in (0, h)$ such that:

$$f(x+h) = f(x) + f'(x)h + \tfrac{1}{2} f''(x+\eta)h^2, \qquad \eta \in (0, h).$$

The idea now is to construct a **necessary** condition for x_* to be the solution of the optimization problem:

$$x_* = \arg\min_{x\in\mathbb{R}} f(x) \tag{2.1}$$

(here, we are taking $S = \mathbb{R}$ for simplicity. As such, we assume that x_* exists, and see what conditions f has to satisfy at x_*.

Theorem 2.2 (First-Order Necessary Condition) *Let x_* be the solution of the OP* (2.1) *(local minimum). Then $f'(x_*) = 0$.*

Proof: Assume for contradiction that $f'(x_*) \neq 0$. Look at the case where $f'(x_*) < 0$. Then by continuity, there is a neighbourhood $I = (x_* - \delta, x_* + \delta)$ (with $\delta > 0$), such that

$$f'(x) < 0, \qquad x \in I.$$

Now consider $x = x_* + h$, where $|h| < \delta$, but h is otherwise not specified. By Taylor's Theorem,

$$f(x) = f(x_*) + f'(x_* + \eta)h, \qquad |\eta| < |h| < \delta.$$

Thus, $f'(x_* + \eta) < 0$. As h is not specified, we can choose it to be $h = -kf'(x_* + \eta)$, where k is a positive constant, such that

$$f(x) = f(x_*) - k|f'(x_* + \eta)|^2,$$

thus, $f(x) < f(x_*)$, which contradicts the fact that $f(x_*)$ is a local minimum. Thus, $f'(x_*) = 0$. ∎

Theorem 2.3 (First-Order necessary condition, $\mathbb{R}^n$) *Let $\boldsymbol{x}_*$ be the solution of the OP*

$$\boldsymbol{x}_* = \arg\min f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \mathbb{R}^n.$$

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

Theorem 2.4 (Second-Order Necessary Condition) *Let x_* be the solution of the OP* (2.1) *(local minimum). Then $f''(x_*) \geq 0$.*

Proof: Again, assume for contradiction that $f''(x_*) < 0$. Then, $f''(x) < 0$ in a neighbourhood $I = (x_* - \delta, x_* + \delta)$ (with $\delta > 0$). By continuity,

$$f''(x) < 0, \qquad x \in I.$$

As before, consider $x = x_* + h$, where $|h| < \delta$ but h is otherwise not specificied. By Taylor's Theorem with $f'(x_*) = 0$, we have;

$$f(x) = f(x_*) + \tfrac{1}{2} f''(x_* + \eta)h^2, \qquad |\eta| < |h| < \delta.$$

Here, $f''(x_* + \eta) < 0$, hence $f(x) < f(x_*)$, which is a contradiction, hence $f''(x_*) \geq 0$. ∎

Theorem 2.5 (Second-Order necessary condition, $\mathbb{R}^n$) *Let $\boldsymbol{x}_*$ be the solution of the OP*

$$\boldsymbol{x}_* = \arg\min f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \mathbb{R}^n.$$

Then the **Hessian matrix** H*, with entries*

$$H_{ij} = \left(\frac{\partial^2 f}{\partial x_i \partial x_j} \right)_{\boldsymbol{x}_*}$$

is positive semi-definite.

We now define **sufficient conditions** on $f(\boldsymbol{x}_*)$ which guarantee that $\boldsymbol{x}_*$ is a local minimizer. Again, we prove the result in 1D and then extend by analogy to higher dimensions.

Theorem 2.6 (Second-Order Sufficient Condition) *Suppose that f is twice differentiable with continuous second derivative, and that $f'(x_*) = 0$ and furthermore, that $f''(x_*) > 0$ (strict). Then x_* is a strict local minimizer of f.*

Proof: Because $f''(x_*) > 0$, by continuity, there is an interval $I = (x_* - \delta, x_* + \delta)$ (with $\delta > 0$), such that $f''(x) > 0$ for all $x \in I$. For each $x \in I$, we write $x = x_* + h$, where $|h| < \delta$, and we have:

$$f(x) = f(x_*) + \tfrac{1}{2} f''(x_* + \eta)h^2, \qquad |\eta| < |h| < \delta.$$

Here, $f''(x_* + \eta) > 0$, hence, $f(x) > f(x_*)$ for all $x \in I$, hence, x_* is a strict local minimizer. ∎

Theorem 2.7 (Second-Order sufficient condition, $\mathbb{R}^n$) *Suppose that f is twice differentiable with continuous second derivative, and that $\nabla f(\boldsymbol{x}_*) = 0$ and furthermore, that the Hessian matrix H is positive-definite. Then $\boldsymbol{x}_*$ is a strict local minimizer of f.*

The reason for breaking up the conditions into necessary and sufficient like this is to handle degenerate points – for instance, there are local minima that have $f'(x) = 0$ but $f''(x) = 0$ also, meaning the second-derivative test is inconclusive. E.g. $f(x) = x^4$. However, if we deal with convex functions, we don't have to worry about the second derivative test at all. And we already know from Chapter 2, that global and local minizers are one and the same for such functions. We prove these facts now, going straight to the case where $f : \mathbb{R}^n \to \mathbb{R}$.

Theorem 2.8 *When f is convex, any local minimizer x_* is a global minimizer of f. If, in addition, f is differentiable, then any stationary point x_* is a global minimizer of f.*

The proof comes in two parts. For the first part, we want to show that a local minimizer really is a global minimizer. To do this, assume for contradiction that $\boldsymbol{x}_*$ is a local minimizer but that there is another $\boldsymbol{y} \in \mathbb{R}^n$ that is the global minimizer, thus

$$f(\boldsymbol{y}) < f(\boldsymbol{x}_*).$$

We now construct a line segment joining these points:

$$\boldsymbol{x}(t) = t\boldsymbol{y} + (1-t)\boldsymbol{x}_*, \qquad t \in [0,1],$$

such that $\boldsymbol{x}(0) = \boldsymbol{x}_*$, and $\boldsymbol{x}(1) = \boldsymbol{y}$. By convexity,

$$\begin{aligned} f(\boldsymbol{x}(t)) &\leq tf(\boldsymbol{y}) + (1-t)f(\boldsymbol{x}_*), \\ &< tf(\boldsymbol{x}_*) + (1-t)f(\boldsymbol{x}_*), \\ &= f(\boldsymbol{x}_*). \end{aligned}$$

Thus,

$$f(\boldsymbol{x}(t)) < f(\boldsymbol{x}_*),$$

and the inequality is strict. Refer to Figure 2.1, and consider the neighbourhood $\mathcal{N}$ around $\boldsymbol{x}_*$. Introduce: Any such neighbourhood will contain a part of the line segment $\boldsymbol{x}(t)$, and we call this

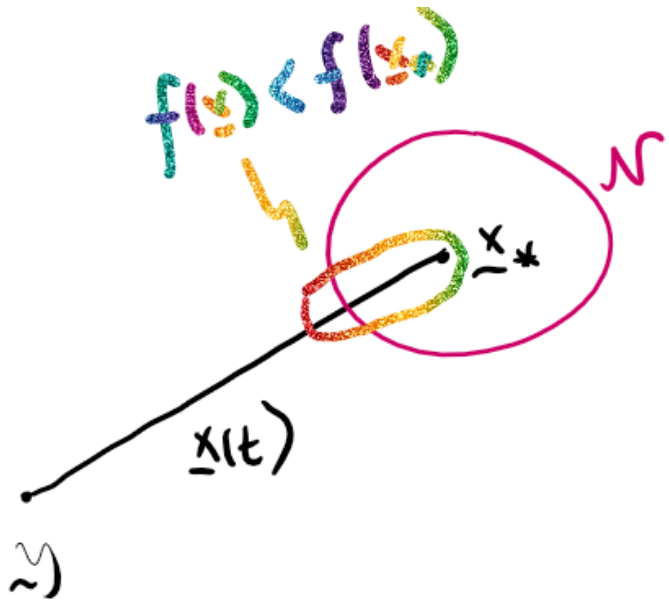


Figure 2.1: A line segment joining two putatively minima

$\mathcal{N}_*$:

$$\mathcal{N}_* = \{\boldsymbol{x}(t) | t \in [0,1]\} \cap \mathcal{N}.$$

Hence:

$$f(\boldsymbol{x}) < f(\boldsymbol{x}_*), \qquad \boldsymbol{x} \in \mathcal{N}_*.$$

which is a contradiction, since $\boldsymbol{x}_*$ is a local minimum. Therefore, we conclude that $\boldsymbol{x}_*$ and $\boldsymbol{y}$ are one and the same, i.e., a global minimizer.

For the second part, we proceed as follows. For any $\boldsymbol{y}$ we have:

$$\begin{aligned}
0 &= (\boldsymbol{y} - \boldsymbol{x}_*) \cdot \underbrace{\nabla f(\boldsymbol{x}_*)}_{=0}, \\
&= \frac{\mathrm{d}}{\mathrm{d}t} f(\boldsymbol{x}_* + t(\boldsymbol{y} - \boldsymbol{x}_*))\big|_{t=0}, \\
&\overset{(*)}{=} \lim_{t \downarrow 0} \frac{f(\boldsymbol{x}_* + t(\boldsymbol{y} - \boldsymbol{x}_*)) - f(\boldsymbol{x}_*)}{t}, \\
&\leq \lim_{t \downarrow 0} \frac{t f(\boldsymbol{y}) + (1-t) f(\boldsymbol{x}_*) - f(\boldsymbol{x}_*)}{t}, \\
&= f(\boldsymbol{y}) - f(\boldsymbol{x}_*),
\end{aligned}$$

hence

$$f(\boldsymbol{x}_*) \leq f(\boldsymbol{y}),$$

for all $\boldsymbol{y}$. Therefore, we conclude that $\boldsymbol{x}_*$ is a global minimizer. ∎

Remark: The limit denoted by the $(*)$ in the above string of relations is a limit from above. The reason for this is that we have to work with $t > 0$, to preserve the direction of the inequalities.

2.3 Model Problem

In unconstrained optimization (especially convex optimization), every differentiable cost function 'locally looks quadratic', that is, in the neighbourhood of the solution $\boldsymbol{x}_* = \arg\min f(\boldsymbol{x})$, the cost function 'looks like a quadratic'. For this reason, quadratic cost functions form an important model problem in optimization, and we study the quadratic problem here:

$$f(\boldsymbol{x}) = c + \langle \boldsymbol{a}, \boldsymbol{x} \rangle + \tfrac{1}{2} \langle \boldsymbol{x}, B\boldsymbol{x} \rangle. \tag{2.2}$$

Here, c is a constant scalar, $\boldsymbol{a}$ is a constant vector in $\mathbb{R}^n$, and B is a constant $n \times n$ matrix; importantly, B is assumed to be **positive definite**.

Theorem 2.9 *The minimizer of Equation* (2.2) *is*

$$\boldsymbol{x}_* = -B^{-1}\boldsymbol{a}.$$

Proof: We re-write the cost function using index notation:

$$f(x_1, \cdots, x_n) = c + a_i x_i + \tfrac{1}{2} B_{ij} x_i x_j,$$

where summation over repeated indices is assumed. The first-order optimality condition is $\partial f / \partial x_k = 0$ for all $k \in \{1, 2, \cdots, n\}$, hence:

$$a_k + \tfrac{1}{2}(B_{ki} + B_{ik}) x_i = 0.$$

As B is symmetric matrix, this becomes:

$$a_k + B_{ik} x_i = 0,$$

hence

$$\boldsymbol{x}_* = -B^{-1}\boldsymbol{a},$$

and the inverse exists since B is positive-definite. The hessian of the cost function is clearly:

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = B_{ij} = \text{Const.}$$

As B is positive-definite, the second-order optimality condition can be used to conclude that $\boldsymbol{x}_*$ is a minimum.

Remark: By substituting $\boldsymbol{x}_* = -B^{-1}\boldsymbol{a}$ back into the cost function, the minimum can be evaluated as:

$$f_{\text{min}} = c - \tfrac{1}{2}\langle \boldsymbol{a}, B^{-1}\boldsymbol{a} \rangle.$$

Notice, if $c \leq 0$, then f_{min} is negative.

2.3.1 When B is no longer positive definite

The model problem can be extended to cases when B is not positive definite: slightly weaker results can be obtained in the case when B is positive semi-definite. This shows the importance of stating assumptions very clearly when formulating and solving an optimization problem. As such, we have the following theorem:

Theorem 2.10 *Let f be the quadratic function defined by*

$$f(\boldsymbol{x}) = c + \langle \boldsymbol{a}, \boldsymbol{x} \rangle + \tfrac{1}{2}\langle \boldsymbol{x}, B\boldsymbol{x} \rangle,$$

where B is any symmetric matrix. Then the following statements are true:

1. *f attains a minimum if and only if B is positive semi-definite and $\boldsymbol{a}$ is in the range of B. If B is positive semi-definite, then every $\boldsymbol{p}$ satisfying $B\boldsymbol{p} = -\boldsymbol{a}$ is a global minimizer of f.*
2. *f has a unique minimizer if and only if B is positive definite.*

We look at Part 1. We start with the "if" statement, that is, we assume that B is positive semi-definite and that $\boldsymbol{a}$ is in the range of B. Then, there exists a $\boldsymbol{x} \in \mathbb{R}^n$ such that $B\boldsymbol{x} = -\boldsymbol{a}$. For all $\boldsymbol{w}$ in $\mathbb{R}^n$, we have:

$$\begin{aligned} f(\boldsymbol{x}+\boldsymbol{w}) &= f + \langle \boldsymbol{a}, \boldsymbol{x}+\boldsymbol{w} \rangle + \tfrac{1}{2}\langle \boldsymbol{x}+\boldsymbol{w}, B(\boldsymbol{x}+\boldsymbol{w}) \rangle, \\ &= f + \langle \boldsymbol{a}, \boldsymbol{x} \rangle + \tfrac{1}{2}\langle \boldsymbol{x}, B\boldsymbol{x} \rangle \\ &\qquad + \langle \boldsymbol{a}, \boldsymbol{w} \rangle + \tfrac{1}{2}\left[\langle \boldsymbol{x}, B\boldsymbol{w} \rangle + \langle \boldsymbol{w}, B\boldsymbol{x} \rangle\right] + \tfrac{1}{2}\langle \boldsymbol{w}, B\boldsymbol{w} \rangle, \\ &= f(\boldsymbol{x}) + \langle \boldsymbol{a}, \boldsymbol{w} \rangle + \tfrac{1}{2}\left[\langle B\boldsymbol{x}, \boldsymbol{w} \rangle - \langle \boldsymbol{w}, \boldsymbol{a} \rangle\right] + \tfrac{1}{2}\langle \boldsymbol{w}, B\boldsymbol{w} \rangle. \end{aligned}$$

Continuing thus, we have:

$$\begin{aligned} f(\boldsymbol{x}+\boldsymbol{w}) &= f(\boldsymbol{x}) + \langle \boldsymbol{a}, \boldsymbol{w} \rangle + \tfrac{1}{2}\left[-\langle \boldsymbol{a}, \boldsymbol{w} \rangle - \langle \boldsymbol{w}, \boldsymbol{a} \rangle\right] + \tfrac{1}{2}\langle \boldsymbol{w}, B\boldsymbol{w} \rangle, \\ &= f(\boldsymbol{x}) + \tfrac{1}{2}\langle \boldsymbol{w}, B\boldsymbol{w} \rangle, \\ &\geq f(\boldsymbol{x}). \end{aligned}$$

Thus, $f(\boldsymbol{x}+\boldsymbol{w}) \geq f(\boldsymbol{x})$ for all $\boldsymbol{w} \in \mathbb{R}^n$, hence $\boldsymbol{x}$ is a minimizer of f.

For the other way around, assume that $\boldsymbol{x}$ is a minimizer of f. Then, by the first-order optimality condition (Theorem 2.3), $\nabla f = 0$, hence $B\boldsymbol{x} + \boldsymbol{a} = 0$, hence $B\boldsymbol{x} = -\boldsymbol{a}$, hence $\boldsymbol{a}$ is in the range of B. Also, by the second-order optimality condition (Theorem 2.4), B is positive-semi-definite.

For Part 2, we start with the "if" statement and assume that B is positive definite. Then, the non-strict inequality in Part 1 are replaced with strict ones, e.g.

$$\begin{aligned} f(\boldsymbol{x}+\boldsymbol{w}) &= f(\boldsymbol{x}) + \langle \boldsymbol{a}, \boldsymbol{w} \rangle + \tfrac{1}{2}\left[-\langle \boldsymbol{a}, \boldsymbol{w} \rangle - \langle \boldsymbol{w}, \boldsymbol{a} \rangle\right] + \tfrac{1}{2}\langle \boldsymbol{w}, B\boldsymbol{w} \rangle, \\ &= f(\boldsymbol{x}) + \tfrac{1}{2}\langle \boldsymbol{w}, B\boldsymbol{w} \rangle, \\ &> f(\boldsymbol{x}). \end{aligned}$$

Hence, $\boldsymbol{x}$ is a strict minimizer. For the "only if" statement, assume that f has a unique minimizer (call it $\boldsymbol{x}$). We are left with the possibility that either B is positive definite, or B is not positive

definite. Assume for contradiction that B is not positive definite. Then we can find a non-zero vector $\boldsymbol{w}$ such that $B\boldsymbol{w} = 0$. Thus

$$f(\boldsymbol{x} + \boldsymbol{w}) = f(\boldsymbol{x}) + \tfrac{1}{2}\langle \boldsymbol{w}, B\boldsymbol{w}\rangle = f(\boldsymbol{x}) + 0,$$

thus, $\boldsymbol{x} + \boldsymbol{w}$ is also a minimizer, which contradicts the uniqueness assumption. Thus, B must be positive definite. ∎

2.4 A note on norms

In the previous example, the usual inner product on $\mathbb{R}^n$ appears in the construction of the quadratic function (2.2), e.g.

$$\langle \boldsymbol{a}, \boldsymbol{x}\rangle = \sum_{i=1}^{n} a_i x_i.$$

This then leads to the equally 'natural' L^2 norm on $\mathbb{R}^n$,

$$\|\boldsymbol{x}\|_2^2 = \langle \boldsymbol{x}, \boldsymbol{x}\rangle = \sum_{i=1}^{n} x_i^2.$$

In this book, we work almost exclusively with the L^2 norm. This is to take advantage of the Cauchy–Schwarz inequality:

$$\langle \boldsymbol{a}, \boldsymbol{x}\rangle \leq \|\boldsymbol{a}\|_2 \|\boldsymbol{x}\|_2. \tag{2.3}$$

However, all norms in a finite-dimensional space are equivalent. That is, if I have two norms $\|\cdot\|_P$ and $\|\cdot\|_Q$, there exist positive constants c_1 and c_2 such that

$$c_1\|\boldsymbol{x}\|_P \leq \|\boldsymbol{x}\|_Q \leq c_2\|\boldsymbol{x}\|_P, \qquad \text{for all } \boldsymbol{x} \in \mathbb{R}^n.$$

For instance, if we look at the L^1-norm

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|,$$

then by Cauchy–Schwarz with $\boldsymbol{x} = (|x_1|, \cdots, |x_n|)$ and $\boldsymbol{a} = (1, 1, \cdots, 1)$, we have:

$$\sum_{i=1}^{n} |x_i| \leq n \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2},$$

hence

$$\|\boldsymbol{x}\|_1 \leq \sqrt{n}\|\boldsymbol{x}\|_2.$$

But it is immediately obvious that e.g. $\|\boldsymbol{x}\|_1^2 \geq \|\boldsymbol{x}\|_2^2$, hence

$$\|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_1 \leq \sqrt{n}\|\boldsymbol{x}\|_2,$$

establishing the equivalence of the L^1-norm and the L^2-norm.

2.5 Take-home Message

Our results so far are based on basic Calculus and Linear Algebra, and they provide the foundations for unconstrained optimization algorithms for smooth cost functions. In one way or another, the continuous optimization algorithms that we will study are based on trying to find a point where ∇f vanishes.

Chapter 3

Line Search Methods

Overview

In this chapter, we look at Line Search methods.

3.1 Preliminaries

Line Search methods refer to a family of different methods to solve the continuous (unconstrained) optimization problem

$$\boldsymbol{x}_* = \arg\,\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}).$$

The methods use information about the gradient of the cost function to construct a sequence of guesses that converges to the solution of the OP. As such, these are **iterative** methods: they start with a 'reasonable' initial guess $\boldsymbol{x}_0$, and a sequence of iterates is produced, $\{\boldsymbol{x}_k\}_{k=0}^{\infty}$, such that $\boldsymbol{x}_k \to \boldsymbol{x}_*$ as $k \to \infty$. In these methods, $\boldsymbol{x}_{k+1}$ is obtained from $\boldsymbol{x}_k$, e.g.

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

where $\boldsymbol{s}_k$ usually depends on $\nabla f(\boldsymbol{x}_k)$ and sometimes on higher derivatives as well. In all of the Line Search Methods, $\boldsymbol{s}_k$ is broken up into a 'step length' α_k and a search direction $\boldsymbol{p}_k$, where $\boldsymbol{p}_k$ is (usually) a unit vector (but not in the Newton method). When $\boldsymbol{p}_k$ is a unit vector (chosen appropriately), then the step length α_k is computed by solving a 1D optimization problem at each iteration step:

$$\alpha_k = \arg\min_{\alpha>0} f(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k). \tag{3.1}$$

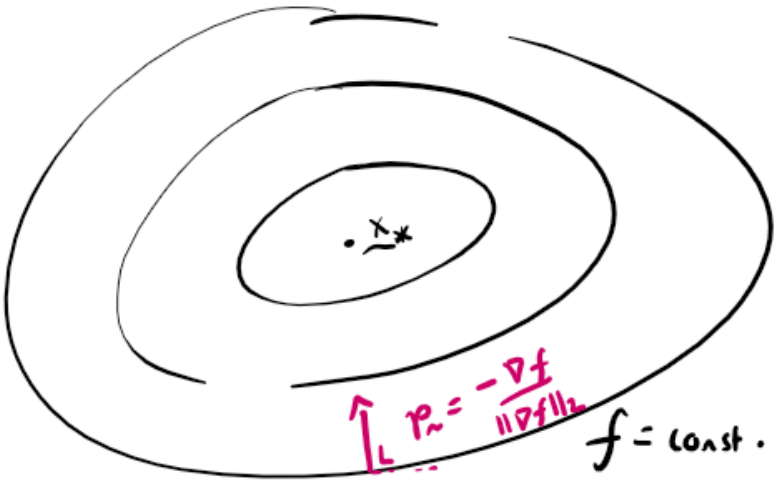


Figure 3.1: Basic idea behind the SD method

3.2 Steepest-Descent Method

We look at the cost function at iteration k. We wish to update $\boldsymbol{x}_k$ by $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha \boldsymbol{p}$, where $\boldsymbol{p}$ is to be determined. By Taylor's theorem (exact), we have:

$$f(\boldsymbol{x}_k + \alpha \boldsymbol{p}) = f(\boldsymbol{x}_k) + \alpha p_i \frac{\partial f}{\partial x_i}(\boldsymbol{x}_k) + \tfrac{1}{2} p_i p_j \alpha^2 \frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{x}_k + t\boldsymbol{p}), \qquad t \in (0, \alpha).$$

The rate of change of f along $\boldsymbol{p}$ is the directional derivative $\boldsymbol{p} \cdot \nabla f(\boldsymbol{x}_k)$. We want to make this change as large and as negative as possible, that way will minimize the cost function and get to $\boldsymbol{x}_*$ as fast as possible. As such, we make $\boldsymbol{p} \cdot \nabla f(\boldsymbol{x}_k)$ as large and as negative as possible, bearing in mind that $\|\boldsymbol{p}\|_2 = 1$. Hence, we take:

$$\boldsymbol{p} = -\frac{\nabla f(\boldsymbol{x}_k)}{\|\nabla f(\boldsymbol{x}_k)\|_2}.$$

This is the direction of **steepest descent** (SD). It is also the direction that is orthogonal to the level sets of $f = \text{Const}$. The idea behind the SD method is shown in Figure 3.1. The basic idea of the algorithm is shown below in Algorithm 1.

Algorithm 1 Outline of SD Method

Choose $\boldsymbol{x}_0$ (sufficiently close to the solution $\boldsymbol{x}_*$).
for $k = 0, 1, 2, \cdots$ **do**
  Compute the steepest descent direction $\boldsymbol{p}_k$.
  Choose the stepsize α_k (somehow, e.g. Equation (5.1))
  Write $\boldsymbol{s}_k = \alpha_k \boldsymbol{p}_k$.
  Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$.
end for

3.3 Newton Method

For the Newton Method, we start with a local **approximation** of $f(\boldsymbol{x})$ at $\boldsymbol{x} = \boldsymbol{x}_k + \boldsymbol{p}$:

$$f(\boldsymbol{x}_k + \boldsymbol{p}) \approx f(\boldsymbol{x}_k) + p_i \frac{\partial f}{\partial x_i}(\boldsymbol{x}_k) + \tfrac{1}{2} p_i p_j \frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{x}_k).$$

Now, we think of $f(\boldsymbol{x}_k)$ as fixed (with $a_i = (\partial f/\partial x_i)(\boldsymbol{x}_k)$ and $B_{ij} = (\partial^2 f/\partial x_i \partial x_j)(\boldsymbol{x}_k)$, and we look at the following function of $\boldsymbol{p}$:

$$m_k(\boldsymbol{p}) = c + a_i p_i + \tfrac{1}{2} p_i p_j B_{ij}$$

Here, $\boldsymbol{p}$ is a variable, and we seek to minimize m_k. We take $\nabla_{\boldsymbol{p}} m_k = 0$ to obtain:

$$a_i = -\sum_{j=1}^{n} B_{ij} p_j.$$

Assuming the matrix B_{ij} is invertible (we actually require more than this), we get:

$$p_i = -\sum_{j=1}^{n} (B^{-1})_{ij} a_j,$$

hence, the descent direction (and step length) is given as:

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

Notation: We use $\boldsymbol{p}_k^N$ for the Newton descent direction, $\boldsymbol{p}_k^N = -B^{-1}(\boldsymbol{x}_k) \nabla f(\boldsymbol{x}_k)$.

Algorithm 2 Outline of the Newton Method

Choose $\boldsymbol{x}_0$ (sufficiently close to the solution $\boldsymbol{x}_*$).

for $k = 0, 1, 2, \cdots$ **do**

Compute the Newton Direction by inverting the Hessian:

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

Notice the stepsize in 'pure' Newton is $\alpha_k = 1$.

Write $\boldsymbol{s}_k = \alpha_k \boldsymbol{p}_k$.

Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$.

end for

3.3.1 Positive-Definite Property

Actually, for the Newton method to work, a sufficient condition is that B be positive-definite: if B is positive-definite, then it is invertible, furthermore,

$$\begin{aligned} f(\boldsymbol{x}_k + t\boldsymbol{p}_k) &= f(\boldsymbol{x}_k) + t\sum_{i=1}^{n}(p_k)_i \frac{\partial f}{\partial x_i}(\boldsymbol{x}_k) + O(t^2), \\ &= f(\boldsymbol{x}_k) - t\sum_{i=1}^{n}\left[\sum_{j=1}^{n}\left(B^{-1}\right)_{ij}\frac{\partial f}{\partial x_j}(\boldsymbol{x}_k)\right]\frac{\partial f}{\partial x_i}(\boldsymbol{x}_k) + O(t^2), \\ &= f(\boldsymbol{x}_k) \underbrace{-t\langle \nabla f(\boldsymbol{x}_k), B^{-1}(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k)\rangle}_{\text{Negative}} + O(t^2), \end{aligned}$$

hence, for t sufficiently small,

$$f(\boldsymbol{x}_k + t\boldsymbol{p}_k) < f(\boldsymbol{x}_k),$$

thus, for t sufficiently small, going off in the direction given by the Newton method reduces f.

3.3.2 Quadratic Convergence

There are several advantages to the Newton step over the standard steepest-descent method:

- Step length is provided, no need to solve the sub-problem 5.1.
- Simple criterion for the method to work (positive-definite Hessian).
- Quadratic convergence

We illustrate what quadratic convergence means in the context of a 1D example, and we look at the general case in $\mathbb{R}^n$ later in Chapter 6. As such, we look at

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}, \tag{3.2}$$

which converges to x_*, with $f'(x_*) = 0$. We let ϵ_k denote the error in the iterative process at iterate k:

$$\begin{aligned} \epsilon_k &= x_* - x_k, \\ \epsilon_{k+1} &= x_* - x_{k+1}. \end{aligned}$$

Substitute into Equation (3.2):

$$x_* - \epsilon_{k+1} = x_* - \epsilon_k - \frac{f'(x_k)}{f''(x_k)}, \tag{3.3}$$

or:

$$\epsilon_{k+1} = \epsilon_k + \frac{f'(x_k)}{f''(x_k)}. \tag{3.4}$$

We have:

$$f'(x_k) = f'(x_* - \epsilon_k) = \cancel{f'(x_*)} - \epsilon_k f''(x_k) + \tfrac{1}{2}\epsilon_k f'''(x_*) + \cdots,$$

and similarly,

$$f''(x_k) = f''(x_* - \epsilon_k) = f''(x_*) - f'''(x_*)\epsilon_k + \cdots$$

We substitute these expansions into Equation (3.4):

$$\begin{aligned} \epsilon_{k+1} &= \epsilon_k + \frac{-\epsilon_k f''(x_k) + \frac{1}{2}\epsilon_k f'''(x_*) + \cdots}{f''(x_*) - f'''(x_*)\epsilon_k + \cdots}, \\ &= \epsilon_k - \epsilon_k \left[\frac{1 - \frac{1}{2}\epsilon_k^2 \frac{f'''(x_k)}{f''(x_k)} + \cdots}{1 - \frac{f'''(x_*)}{f''(x_*)}\epsilon_k + \cdots} \right], \\ &\overset{\text{Binomial Thm}}{=} \epsilon_k - \epsilon_k \left[1 - \tfrac{1}{2}\epsilon_k^2 \frac{f'''(x_k)}{f''(x_k)} + \cdots \right] \left[1 + \frac{f'''(x_*)}{f''(x_*)}\epsilon_k + \cdots \right], \\ &= \epsilon_k - \epsilon_k \left[1 + \frac{f'''(x_*)}{f''(x_*)}\epsilon_k - \tfrac{1}{2}\epsilon_k^2 \frac{f'''(x_*)}{f''(x_*)} + O\left((\epsilon_k)^3\right) \right], \\ &= \cancel{\epsilon_k} - \cancel{\epsilon_k} - \tfrac{1}{2}\epsilon_k^2 \frac{f'''(x_*)}{f''(x_*)} + O\left((\epsilon_k)^3\right) \end{aligned}$$

Hence,

$$\epsilon_{k+1} = -\frac{f'''(x_*)}{2f''(x_*)} \left[\epsilon_k\right]^2 + O\left((\epsilon_k)^3\right),$$

and it is equally valid to write:

$$\epsilon_{k+1} = -\frac{f'''(x_k)}{f''(x_k)} \left[\epsilon_k\right]^2 + O\left((\epsilon_k)^3\right).$$

Thus, we obtain quadratic convergence, e.g. if the error at iterate k is $O(10^{-4})$, then the error at the next iterate is $O(10^{-8})$, and then at the next level $O(10^{-16})$, which is machine precision in three steps.

There are also drawbacks to the Newton method:

- Requires computation of the Hessian at each iteration.
- Requires inversion of the Hessian at each iteration $O(n^3)$.

3.4 Secant Methods

To address the drawbacks of the Newton method, people use **secant methods**. These can be easily understood in the context of the 1D problem, by Taylor expansion:

$$f'(x_k + \delta x) \approx f'(x_k) + f''(x_k)\delta x.$$

If we take $\delta x = x_{k+1} - x_k$, this gives

$$f'(x_{k+1}) - f'(x_k) \approx f''(x_k)[x_{k+1} - x_k]. \tag{3.5}$$

Call

$$y_k = f'(x_{k+1}) - f'(x_k), \qquad s_k = x_{k+1} - x_k.$$

Thus, a numerical approximation for the 1D Hessian is:

$$f''(x_k) \approx y_k / s_k.$$

In n dimensions, the equivalent expression to Equation (3.5) is:

$$\nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k) \approx B(\boldsymbol{x}_k)\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right),$$

or

$$\boldsymbol{y}_k \approx B(\boldsymbol{x}_k)\boldsymbol{s}_k.$$

We write this more precisely as:

$$\boldsymbol{y}_k = B_{k+1}\boldsymbol{s}_k, \tag{3.6}$$

where the matrix B_{k+1} is the approximation to the Hessian.

Algorithm 3 Outline of Secant Method

Choose $\boldsymbol{x}_0$ (sufficiently close to the solution $\boldsymbol{x}_*$).

for $k = 0, 1, 2, \cdots$ **do**

Compute the descent direction by solving $B_k \boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k)$ (somehow!).

Choose the stepsize α_k (somehow!).

Write $\boldsymbol{s}_k = \alpha_k \boldsymbol{p}_k$.

Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{s}_k$.

Update $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$ and update B_{k+1}.

end for

Chapter 4

The BFGS and Barzilai–Borwein Methods

Overview

In this chapter, we study the BFGS formula in detail. Towards the end of the chapter, we also showcase another simple method called the Barzilai–Borwein method.

4.1 Review

We start with a review of the Secant Method from Chapter 3. For the 1D problem where we wish to solve $f'(x_*) = 0$, we construct a sequence of iterates, where

$$f'(x_k + \delta x) \approx f'(x_k) + f''(x_k)\delta x.$$

If we take $\delta x = x_{k+1} - x_k$, this gives

$$f'(x_{k+1}) - f'(x_k) \approx f''(x_k)[x_{k+1} - x_k]. \tag{4.1}$$

Call

$$y_k = f'(x_{k+1}) - f'(x_k), \qquad s_k = x_{k+1} - x_k.$$

Thus, a numerical approximation for the 1D Hessian is:

$$f''(x_k) \approx y_k / s_k.$$

In n dimensions, the equivalent expression to Equation (4.1) is:

$$\nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k) \approx B(\boldsymbol{x}_k)\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right),$$

or

$$\boldsymbol{y}_k \approx B(\boldsymbol{x}_k)\boldsymbol{s}_k.$$

We write this more precisely as:

$$\boldsymbol{y}_k = B_{k+1}\boldsymbol{s}_k, \tag{4.2}$$

4.2 BFGS Formula

Equation (4.2) is an equation for the unknown matrix B_{k+1}. There are $n \times n$ unknowns and only n equations, so there can be a solution only in some approximate sense. In the BFGS formula, B_{k+1} is approximated by B_k and two matrices, built out of the only thing we have:

$$\boldsymbol{y}_k, \qquad B_k\boldsymbol{s}_k.$$

These are $n \times 1$ column vectors. So we build the new matrix as follows:

$$B_{k+1} = B_k + \alpha \boldsymbol{y}_k\boldsymbol{y}_k^T + \beta(B_k\boldsymbol{s}_k)(B_k\boldsymbol{s}_k)^T, \tag{4.3}$$

where now $\boldsymbol{y}_k\boldsymbol{y}_k^T$ is an array of size $(n\times 1)\times(1\times n) = n\times n$, and the same for $(B_k\boldsymbol{s}_k)(B_k\boldsymbol{s}_k)^T$. Here also, α and β are scalars to be determined. We do this simply by imposing the secant condition (4.2).

Theorem 4.1 *The following are the values of α and β in the BFGS method:*

$$\beta = -\frac{1}{\langle \boldsymbol{s}_k, B_k^T\boldsymbol{s}_k\rangle}, \qquad \alpha = \frac{1}{\langle \boldsymbol{y}_k, \boldsymbol{s}_k\rangle}.$$

Proof: The proof is by direct computation. We take the approximation (4.3) for B_{k+1} and we impose the secant condition (4.2). This gives:

$$\begin{aligned}
\boldsymbol{y}_k &= B_{k+1}\boldsymbol{s}_k, \\
&= \left[B_k + \alpha\boldsymbol{y}_k\boldsymbol{y}_k^T + \beta(B_k\boldsymbol{s}_k)(B_k\boldsymbol{s}_k)^T\right]\boldsymbol{s}_k, \\
&= B_k\boldsymbol{s}_k + \alpha\boldsymbol{y}_k\boldsymbol{y}_k^T\boldsymbol{s}_k + \beta B_k\boldsymbol{s}_k\boldsymbol{s}_k^TB_k^T\boldsymbol{s}_k, \\
0 &= B_k\boldsymbol{s}_k + \beta(B_k\boldsymbol{s}_k)\underbrace{\boldsymbol{s}_k^TB_k^T\boldsymbol{s}_k}_{(1\times n)(n\times n)(n\times 1)=1\times 1} + \alpha\boldsymbol{y}_k\underbrace{(\boldsymbol{y}_k^T\boldsymbol{s}_k)}_{(1\times n)(n\times 1)=1\times 1} - \boldsymbol{y}_k.
\end{aligned}$$

Hence,

$$0 = B_k\boldsymbol{s}_k\left(1 + \beta\langle \boldsymbol{s}_k, B_k^T\boldsymbol{s}_k\rangle\right) + \boldsymbol{y}_k\left(-1 + \alpha\langle \boldsymbol{y}_k, \boldsymbol{s}_k\rangle\right),$$

and hence finally,

$$\beta = -\frac{1}{\langle \boldsymbol{s}_k, B_k^T\boldsymbol{s}_k\rangle}, \qquad \alpha = \frac{1}{\langle \boldsymbol{y}_k, \boldsymbol{s}_k\rangle}. \quad \blacksquare$$

We put all of these calculations together to obtain a final formula for the approximate Hessian in the BFGS method:

$$B_{k+1} = B_k - \frac{B_k \boldsymbol{s}_k s_k^T B_k}{\langle \boldsymbol{s}_k, B_k^T \boldsymbol{s}_k \rangle} + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\langle \boldsymbol{y}_k, \boldsymbol{s}_k \rangle}. \tag{4.4}$$

4.2.1 Positive-Definite Property

The BFGS method has the desirable property that the positive-definiteness of B_k is maintained, which means that $\boldsymbol{p}_k$ is a search direction that always minimizes the cost function. We prove this as follows.

Theorem 4.2 *Provided B_0 is positive definite and provided the curvature condition $\langle \boldsymbol{y}_k, \boldsymbol{s}_k \rangle$ is satisfied for each iteration, the BFGS method produces a symmetric positive-definite approximation to the Hessian at each iteration.*

We start with symmetry:

$$B_{k+1} = B_k + \text{Symmetric Matrices},$$

if B_0 is symmetric, then by induction, B_k is symmetric for all $k \in \{1, 2, \cdots\}$.

We next look at positive definiteness. Again, we have to proceed by mathematical induction, so we assume that B_k is positive definite. We then show that B_{k+1} is positive definite. We also use the extra assumption of positive curvature $\langle \boldsymbol{y}_k, \boldsymbol{s}_k \rangle > 0$ at each iteration. As such, for any vector $\boldsymbol{\xi} \in \mathbb{R}^n$, we have:

$$\begin{aligned} \langle \boldsymbol{\xi}, B_{k+1} \boldsymbol{\xi} \rangle &= \langle \boldsymbol{\xi}, B_k \boldsymbol{\xi} \rangle - \frac{\langle \boldsymbol{\xi}, B_k \boldsymbol{s}_k \boldsymbol{s}_k^T B_k \boldsymbol{\xi} \rangle}{\langle \boldsymbol{s}_k, B_k^T \boldsymbol{s}_k \rangle} + \frac{\langle \boldsymbol{\xi}, \boldsymbol{y}_k \boldsymbol{y}_k^T \boldsymbol{\xi} \rangle}{\langle \boldsymbol{y}_k, \boldsymbol{s}_k \rangle}, \\ &= \langle \boldsymbol{\xi}, B_k \boldsymbol{\xi} \rangle - \frac{\langle \boldsymbol{\xi}, B_k \boldsymbol{s}_k \boldsymbol{s}_k^T B_k \boldsymbol{\xi} \rangle}{\langle \boldsymbol{s}_k, B_k^T \boldsymbol{s}_k \rangle} + \frac{\langle \boldsymbol{\xi}, \boldsymbol{y}_k \rangle^2}{\langle \boldsymbol{y}_k, \boldsymbol{s}_k \rangle}. \end{aligned}$$

As the second term is positive definite, we focus on the first term, we call it Δ, and so we have to show that $\Delta > 0$:

$$\Delta = \langle \boldsymbol{\xi}, B_k \boldsymbol{\xi} \rangle - \frac{\langle \boldsymbol{\xi}, B_k \boldsymbol{s}_k \boldsymbol{s}_k^T B_k \boldsymbol{\xi} \rangle}{\langle \boldsymbol{s}_k, B_k^T \boldsymbol{s}_k \rangle}.$$

By induction B_k is positive definite, so it is diagonalizable, with positive eigenvalues λ_i and eigenvectors $\boldsymbol{u}_i$. As such, we have

$$\boldsymbol{\xi} = \sum_i \xi_i \boldsymbol{u}_i, \qquad \xi_i = \langle \boldsymbol{\xi}, \boldsymbol{u}_i \rangle, \qquad \boldsymbol{s}_k = \sum_i \sigma_i \boldsymbol{u}_i, \qquad \sigma_i = \langle \boldsymbol{s}_k, \boldsymbol{u}_i \rangle.$$

and

$$\langle \boldsymbol{\xi}, B_k \boldsymbol{\xi} \rangle = \sum_i \lambda_i \xi_i^2.$$

Hence,

$$\begin{aligned}
\Delta &= \sum_i \lambda_i \xi_i^2 - \frac{\langle B_k\boldsymbol{\xi}, \boldsymbol{s}_k\boldsymbol{s}_k^T B_k\boldsymbol{\xi}\rangle}{\sum_i \lambda_i\sigma_i^2}, \\
&= \sum_i \lambda_i \xi_i^2 - \frac{\langle \sum_i \lambda_i\xi_i\boldsymbol{u}_i, (\boldsymbol{s}_k\boldsymbol{s}_k^T)\sum_j \lambda_j\xi_j\boldsymbol{u}_j\rangle}{\sum_i \lambda_i\sigma_i^2}, \\
&= \sum_i \lambda_i \xi_i^2 - \frac{\sum_i\sum_j \lambda_i\lambda_j\xi_j\xi_i\langle \boldsymbol{u}_i, (\boldsymbol{s}_k\boldsymbol{s}_k^T)\boldsymbol{u}_j\rangle}{\sum_i \lambda_i\sigma_i^2}, \\
&= \sum_i \lambda_i \xi_i^2 - \frac{\sum_i\sum_j \lambda_i\lambda_j\xi_j\xi_i\sigma_i\sigma_j}{\sum_i \lambda_i\sigma_i^2}, \\
&= \frac{\sum_i \lambda_i\xi_i^2 \sum_j \lambda_j\sigma_j^2 - \sum_i\sum_j \lambda_i\lambda_j\xi_j\xi_i\sigma_i\sigma_j}{\sum_j \lambda_j\sigma_j^2},
\end{aligned}$$

If we introduce two new vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$, with components $X_i = \sqrt{\lambda_i}\xi_i$ and $Y_i = \sqrt{\lambda_i}\sigma_i$, then Δ can be re-written as:

$$\Delta = \frac{\|\boldsymbol{X}\|_2^2\|\boldsymbol{Y}\|_2^2 - (\boldsymbol{X}\cdot\boldsymbol{Y})^2}{\|\boldsymbol{Y}\|_2^2},$$

which is positive, by Cauchy-Schwarz. Hence, B_{k+1} is positive definite.

Furthermore, as B_0 is assumed to be positive definite, by mathematical induction, B_k is positive definite, for all $k \in \{0, 1, 2, \cdots\}$. ∎

4.2.2 Sherman-Morrison–Woodbury Formula

In reality, we are not that interested in B_k, only in B_k^{-1}. Therefore, the BFGS method will simplify if we can find a simple way to compute B_k^{-1} directly at each iteration. More precisely, a performing a matrix inversion is $O(n^3)$, however, if we can find a way of exploiting the structure of the matrix B_k, then maybe this can be reduced. The key here is to recall the Sherman-Morrison–Woodbury formula from Linear Algebra:

Theorem 4.3 (Sherman–Morrison–Woodbury) *Given a square invertible $n \times n$ matrix M, an $n \times k$ matrix U, and a $k \times n$ matrix V, let M be an $n \times n$ matrix such that $M = B + UV$. Then, assuming $(\mathbb{I}_k + VB^{-1}U)$ is invertible, we have:*

$$M^{-1} = B^{-1} - B^{-1}U\left(\mathbb{I}_k + VB^{-1}U\right)^{-1}VB^{-1}. \tag{4.5}$$

For our purposes, we recall the BFGS formula (4.4), recalled here as:

$$B_{k+1} = B_k - \frac{B_k\boldsymbol{s}_k s_k^T B_k}{\langle \boldsymbol{s}_k, B_k^T\boldsymbol{s}_k\rangle} + \frac{\boldsymbol{y}_k\boldsymbol{y}_k^T}{\langle \boldsymbol{y}_k, \boldsymbol{s}_k\rangle}.$$

We are going to suppress the subscripts, and write this as:

$$M = B - \frac{B\boldsymbol{s}s^T B}{\langle \boldsymbol{s}, B^T\boldsymbol{s}\rangle} + \frac{\boldsymbol{y}\boldsymbol{y}^T}{\langle \boldsymbol{y}, \boldsymbol{s}\rangle}.$$

We match this up with Equation (4.5) as follows:

$$M = B + \underbrace{[B\boldsymbol{s}, \boldsymbol{y}]}_{=U,\, 2\times n} \underbrace{\begin{bmatrix} -\frac{1}{\langle \boldsymbol{s}, B\boldsymbol{s}\rangle} & 0 \\ 0 & \frac{1}{\langle \boldsymbol{y}, \boldsymbol{s}\rangle} \end{bmatrix} \begin{bmatrix} \boldsymbol{s}^T B \\ \boldsymbol{y}^T \end{bmatrix}}_{=V^T,\, 2\times n}$$

Thus, $k = 2$, and the Sherman–Morrisson–Woodbury formula tells us that:

$$M^{-1} = B^{-1} - B^{-1}U\left(\mathbb{I}_2 + VB^{-1}U\right)^{-1} VB^{-1}.$$

So we now go through the calculations:

$$\begin{aligned} M^{-1} &= B^{-1} \\ &- B^{-1}\left[B\boldsymbol{s}, \boldsymbol{y}\right]\left(\mathbb{I}_2 + \begin{bmatrix} -\frac{1}{\langle \boldsymbol{s}, B\boldsymbol{s}\rangle} & 0 \\ 0 & \frac{1}{\langle \boldsymbol{y}, \boldsymbol{s}\rangle} \end{bmatrix} \begin{bmatrix} \boldsymbol{s}^T B \\ \boldsymbol{y}^T \end{bmatrix} B^{-1}[B\boldsymbol{s}, \boldsymbol{y}]\right)^{-1} \begin{bmatrix} -\frac{1}{\langle \boldsymbol{s}, B\boldsymbol{s}\rangle} & 0 \\ 0 & \frac{1}{\langle \boldsymbol{y}, \boldsymbol{s}\rangle} \end{bmatrix} \begin{bmatrix} \boldsymbol{s}^T B \\ \boldsymbol{y}^T \end{bmatrix} B^{-1} \end{aligned}$$

To make the notation a bit clearer we use $H = B^{-1}$:

$$\begin{aligned} M^{-1} &= H^{-1} \\ &- H\left[B\boldsymbol{s}, \boldsymbol{y}\right]\left(\mathbb{I}_2 + \begin{bmatrix} -\frac{1}{\langle \boldsymbol{s}, B\boldsymbol{s}\rangle} & 0 \\ 0 & \frac{1}{\langle \boldsymbol{y}, \boldsymbol{s}\rangle} \end{bmatrix} \begin{bmatrix} \boldsymbol{s}^T B \\ \boldsymbol{y}^T \end{bmatrix} H[B\boldsymbol{s}, \boldsymbol{y}]\right)^{-1} \begin{bmatrix} -\frac{1}{\langle \boldsymbol{s}, B\boldsymbol{s}\rangle} & 0 \\ 0 & \frac{1}{\langle \boldsymbol{y}, \boldsymbol{s}\rangle} \end{bmatrix} \begin{bmatrix} \boldsymbol{s}^T B \\ \boldsymbol{y}^T \end{bmatrix} H \end{aligned}$$

We multiply in by the H on the left and the H on the right-hand side, using $BH = HB = \mathbb{I}_n$:

$$\begin{aligned} M^{-1} &= H^{-1} \\ &- \left[\boldsymbol{s}, H\boldsymbol{y}\right]\left(\mathbb{I}_2 + \begin{bmatrix} -\frac{1}{\langle \boldsymbol{s}, B\boldsymbol{s}\rangle} & 0 \\ 0 & \frac{1}{\langle \boldsymbol{y}, \boldsymbol{s}\rangle} \end{bmatrix} \begin{bmatrix} \boldsymbol{s}^T B \\ \boldsymbol{y}^T \end{bmatrix} H[B\boldsymbol{s}, \boldsymbol{y}]\right)^{-1} \begin{bmatrix} -\frac{1}{\langle \boldsymbol{s}, B\boldsymbol{s}\rangle} & 0 \\ 0 & \frac{1}{\langle \boldsymbol{y}, \boldsymbol{s}\rangle} \end{bmatrix} \begin{bmatrix} \boldsymbol{s}^T \\ \boldsymbol{y}^T H \end{bmatrix} \end{aligned}$$

Let's call:

$$D = \begin{bmatrix} -\frac{1}{\langle \boldsymbol{s}, B\boldsymbol{s}\rangle} & 0 \\ 0 & \frac{1}{\langle \boldsymbol{y}, \boldsymbol{s}\rangle} \end{bmatrix}.$$

We bring D inside the $(\mathbb{I}_2 + \cdots)^{-1}D$ matrix term, and remember that the inverse of a product is the product of the inverse, in reverse order, hence $(\mathbb{I}_2 + \cdots)^{-1}D = [D^{-1} + D^{-1}(\cdots)]^{-1}$, hence:

$$\begin{aligned}
M^{-1} &= H^{-1} - [\boldsymbol{s}, H\boldsymbol{y}] \left(\begin{bmatrix} -\langle \boldsymbol{s}, B\boldsymbol{s} \rangle & 0 \\ 0 & \langle \boldsymbol{y}, \boldsymbol{s} \rangle \end{bmatrix} + \begin{bmatrix} \boldsymbol{s}^T B \\ \boldsymbol{y}^T \end{bmatrix} H[B\boldsymbol{s}, \boldsymbol{y}] \right)^{-1} \begin{bmatrix} \boldsymbol{s}^T \\ \boldsymbol{y}^T H \end{bmatrix}, \\
&= H^{-1} - [\boldsymbol{s}, H\boldsymbol{y}] \left(\begin{bmatrix} -\langle \boldsymbol{s}, B\boldsymbol{s} \rangle & 0 \\ 0 & \langle \boldsymbol{y}, \boldsymbol{s} \rangle \end{bmatrix} + \begin{bmatrix} \boldsymbol{s}^T B \\ \boldsymbol{y}^T \end{bmatrix} [\boldsymbol{s}, H\boldsymbol{y}] \right)^{-1} \begin{bmatrix} \boldsymbol{s}^T \\ \boldsymbol{y}^T H \end{bmatrix}, \\
&= H^{-1} - [\boldsymbol{s}, H\boldsymbol{y}] \left(\begin{bmatrix} -\langle \boldsymbol{s}, B\boldsymbol{s} \rangle & 0 \\ 0 & \langle \boldsymbol{y}, \boldsymbol{s} \rangle \end{bmatrix} + \begin{bmatrix} \langle \boldsymbol{s}, B\boldsymbol{s} \rangle & \langle \boldsymbol{s}, \boldsymbol{y} \rangle \\ \langle \boldsymbol{s}, \boldsymbol{y} \rangle & \langle \boldsymbol{y}, H\boldsymbol{y} \rangle \end{bmatrix} \right)^{-1} \begin{bmatrix} \boldsymbol{s}^T \\ \boldsymbol{y}^T H \end{bmatrix}, \\
&= H^{-1} - [\boldsymbol{s}, H\boldsymbol{y}] \left(\begin{bmatrix} -0 & \langle \boldsymbol{s}, \boldsymbol{y} \rangle \\ \langle \boldsymbol{s}, \boldsymbol{y} \rangle & \langle \boldsymbol{y}, \boldsymbol{s} \rangle + \langle \boldsymbol{y}, H\boldsymbol{y} \rangle \end{bmatrix} \right)^{-1} \begin{bmatrix} \boldsymbol{s}^T \\ \boldsymbol{y}^T H \end{bmatrix}, \\
&= H^{-1} - [\boldsymbol{s}, H\boldsymbol{y}] \left(\begin{bmatrix} 0 & \langle \boldsymbol{s}, \boldsymbol{y} \rangle \\ \langle \boldsymbol{s}, \boldsymbol{y} \rangle & \langle \boldsymbol{y}, \boldsymbol{s} \rangle + \langle \boldsymbol{y}, H\boldsymbol{y} \rangle \end{bmatrix} \right)^{-1} \begin{bmatrix} \boldsymbol{s}^T \\ \boldsymbol{y}^T H \end{bmatrix}, \\
&= H^{-1} + \frac{[\boldsymbol{s}, H\boldsymbol{y}]}{\langle \boldsymbol{s}, \boldsymbol{y} \rangle^2} \begin{bmatrix} \langle \boldsymbol{y}, \boldsymbol{s} \rangle + \langle \boldsymbol{y}, H\boldsymbol{y} \rangle & -\langle \boldsymbol{s}, \boldsymbol{y} \rangle \\ -\langle \boldsymbol{s}, \boldsymbol{y} \rangle & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{s}^T \\ \boldsymbol{y}^T H \end{bmatrix}, \\
&= H^{-1} + \left(1 + \frac{\langle \boldsymbol{y}, H\boldsymbol{y} \rangle}{\langle \boldsymbol{s}, \boldsymbol{y}} \right) \frac{\boldsymbol{s}\boldsymbol{s}^T}{\langle \boldsymbol{s}, \boldsymbol{y} \rangle} - \frac{H\boldsymbol{y}\boldsymbol{s}^T + (H\boldsymbol{y}\boldsymbol{s}^T)^T}{\langle \boldsymbol{s}, \boldsymbol{y} \rangle^2}.
\end{aligned}$$

Restoring the superscripts, the update rule for the BFGS method is:

$$B_{k+1}^{-1} = B_k^{-1} + \left(1 + \frac{\langle \boldsymbol{y}_k, B_k^{-1}\boldsymbol{y}_k \rangle}{\langle \boldsymbol{s}_k, \boldsymbol{y}_k \rangle} \right) \frac{\boldsymbol{s}_k(\boldsymbol{s}_k)^T}{\langle \boldsymbol{s}_k, \boldsymbol{y}_k \rangle} - \frac{B_k^{-1}\boldsymbol{y}_k(\boldsymbol{s}_k)^T + \boldsymbol{s}_k(\boldsymbol{y}_k)^T B_k^{-1}}{\langle \boldsymbol{s}_k, \boldsymbol{y}_k \rangle}. \tag{4.6}$$

Thus, we can work with B_k^{-1} and B_{k+1}^{-1} directly, without ever having to know B_k or B_{k+1}.

4.2.3 Operation Count

Equation (4.6) involves updating each entry in an $n \times n$ matrix, hence the operation count is $O(n^2)$. This is the limiting factor in the iteration step in Algorithm 3 ('Outline of Secant Method'), as the other steps are $O(n)$. Thus, the operation count for the BFGS method is $O(n^2)$. This is a significant speedup compared to the ordinary Newton method, which is $O(n^3)$ (the limiting factor there is Gaussian elimination at every iteration).

4.3 Barzilai–Borwein Formula

In the BFGS formula, the idea is to update B_k using a rank-two matrix. Barzilai–Borwein is much simpler. Instead, we approximate B_k at each iteration by a diagonal matrix:

$$\boldsymbol{s}_k = \boldsymbol{x}_k - \boldsymbol{x}_{-1}, \qquad \boldsymbol{y}_k = \nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_{k-1}),$$

with

$$\boldsymbol{y}_k = B_k \boldsymbol{s}_k$$

Now, the matrix B_k is approximated as a simple diagonal matrix:

$$B_k \approx \frac{1}{\alpha_k} \mathbb{I}_n.$$

The equation $\boldsymbol{y}_k = B_k \boldsymbol{s}_k$ is then solved in the least-squares sense:

$$\alpha_k = \arg\min_{\alpha>0} \| \frac{1}{\alpha} \boldsymbol{s}_k - \boldsymbol{y}_k \|_2^2,$$

this has an analytical solution

$$\alpha_k = \frac{\|\boldsymbol{s}_k\|_2^2}{\langle \boldsymbol{s}_k, \boldsymbol{y}_k \rangle}.$$

The update step is thus:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - B_k^{-1} \nabla f(\boldsymbol{x}^k), \\ &= \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}^k). \end{aligned}$$

Advantages:

- Fast and simple, no need to use Equation (5.1) to estimate step size α_k.

Disadvantages:

- Convergence is not guaranteed to be monotonic, i.e. $f(\boldsymbol{x}_{k+1})$ is not less than $f(\boldsymbol{x}_k)$ necessarily.
- Notice how the iterates are constructed – the method is not self-starting, another method is needed for $k = 0$.

Chapter 5

Line-Search Methods – Stepsize Analysis

Overview

In Chapter 3, we showed how the Line Search methods involves a sub-problem in which it is required to compute the optimal step size α_k, for a given descent direction $\boldsymbol{p}_k$.

$$\alpha_k = \arg\min_{\alpha>0} f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k). \tag{5.1}$$

This is yet another optimization problem, albeit a 1D problem. In this chapter we look at quick alternatives to solving the full problem (5.1).

5.1 Strong Wolfe Conditions

We view

$$\phi(\alpha) = f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k)$$

as a simple one-dimensional problem, which we seek to minimize. However, instead of minimizing α, we can chose an 'okay' value of α that gets us close to the minimum. The rationale here is that what we really wish to minimize is $\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x})$, so how close we are to the minimum of the sub-problem (5.1) should not really matter, so long as we are 'close enough'.

As such, consider a linear approximation to $\phi(\alpha)$:

$$\phi(\alpha) \approx \phi(0) + \phi'(0)\alpha,$$

where $\phi'(0) = \boldsymbol{p}_k \cdot \nabla f(\boldsymbol{x}_k)$, which we assume is negative, since $\boldsymbol{p}_k$ is supposed to be taking us down towards the minimum. Thus, $\phi(\alpha)$ should look something like the graph in Figure 5.1.

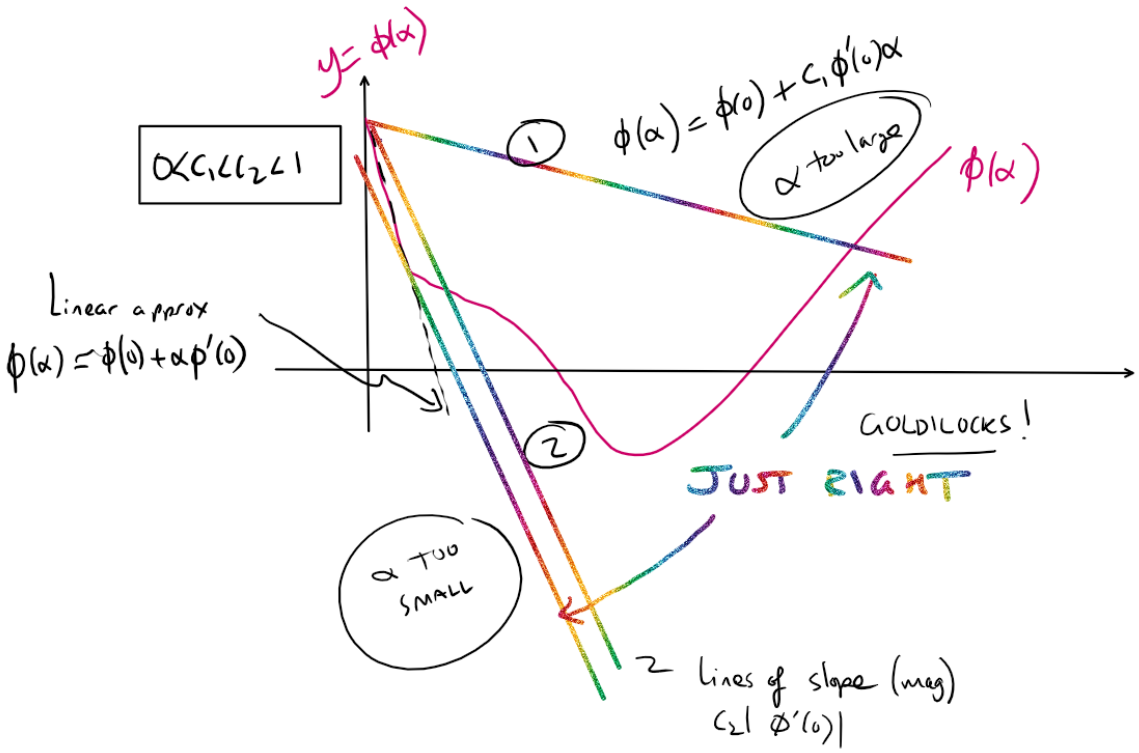


Figure 5.1: Idea behind the Wolfe conditions

5.1.1 Upper cutoff

Fix a value $c_1 \in (0, 1)$. Now, suppose my guess for the stepsize α is too large. Then, the value of $\phi(\alpha)$ is above Line 1 in the figure, hence

$$\phi(\alpha) > \phi(0) + c_1\phi'(0)\alpha,$$

where here c_1 is the slope parameter of Line 1, chosen such that Line 1 has a shallower slope than the linear approximation $\phi(\alpha) \approx \phi(0) + \phi'(0)\alpha$. So clearly, to be anywhere near the local minimum, I require:

$$\phi(\alpha) \leq \phi(0) + c_1\phi'(0)\alpha. \tag{5.2}$$

5.1.2 Lower cutoff

Notice that even $\alpha = 0$ will satisfy Equation (5.2). Obviously, $\alpha = 0$ is no good, because I will then fail to make any progress with the iterative method. So $\alpha = 0$ should be exculded – as well as other 'very small' values of α. A way to exclude very small values of α can be reasoned out as follows. Fix another value $c_2 \in (0, 1)$, and suppose my guess for the stepsize α is too small. Then, the value of the **slope** of $\phi(\alpha)$ will be less than the slope of Line 2 in the figure, in other words, the

slope $\phi'(\alpha)$ is unacceptably steep. Since $\phi'(\alpha) < 0$, 'unacceptably steep' means:

$$|\phi'(\alpha)| > c_2|\phi'(0)|,$$

so an acceptable slope is thus:

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

5.1.3 Combination of both cutoffs

We combine the two cutoff criteria in one place, these are called the **Strong Wolfe Conditions** (SWCs). We state these conditions now as follows, these conditions are valid for fixed $c_1 \in (0,1)$ and fixed $c_2 \in (0,1)$, actually we require $0 < c_1 < c_2 < 1$.

$$\begin{aligned} \phi(\alpha) &\leq \phi(0) + c_1\phi'(0)\alpha, && \text{(5.3a)} \\ |\phi'(\alpha)| &\leq c_2|\phi'(0)|. && \text{(5.3b)} \end{aligned}$$

We show the necessity for the ordering condition $0 < c_1 < c_2 < 1$ in the next subsection.

Nomenclature: SW1 is also referred to as the **Armijo** condition.

5.1.4 Existence Criterion

Theorem 5.1 (Strong Wolfe Conditions) *Let $\phi(\alpha)$ be a continuously differentiable function which is bounded below, $\phi(\alpha) \geq \phi_{min}$. If $0 < c_1 < c_2 < 1$, then there exists an $\alpha > 0$ satisfying the SWCs.*

Fix $0 < c_1 < 1$, and consider

$$\begin{aligned} \Delta(\alpha) &= \ell(\alpha) - \phi(\alpha), \\ &= [\phi(0) + c_1\alpha\phi'(0)] - \phi(\alpha). \end{aligned}$$

We have $\phi(\alpha) \geq \phi_{min}$, hence $-\phi(\alpha) \leq -\phi_{min}$, hence

$$\Delta \leq \phi(0) + \phi'(0)c_1\alpha - \phi_{min}.$$

As $\phi'(0) < 0$ (by choice of $\boldsymbol{p}_k$), we have $\Delta \to -\infty$ as $\alpha \to \infty$. Furthermore, $\Delta(0) = 0$, and

$$\begin{aligned} \Delta'(0) &= c_1\phi'(0) - \phi'(0), \\ &= (c_1 - 1)\phi'(0), \\ &> 0, \end{aligned}$$

since $0 < c_1 < 1$ and $\phi'(0) < 0$. Putting all of this information together, the graph of $\Delta(\alpha)$ looks something like Figure 5.2.

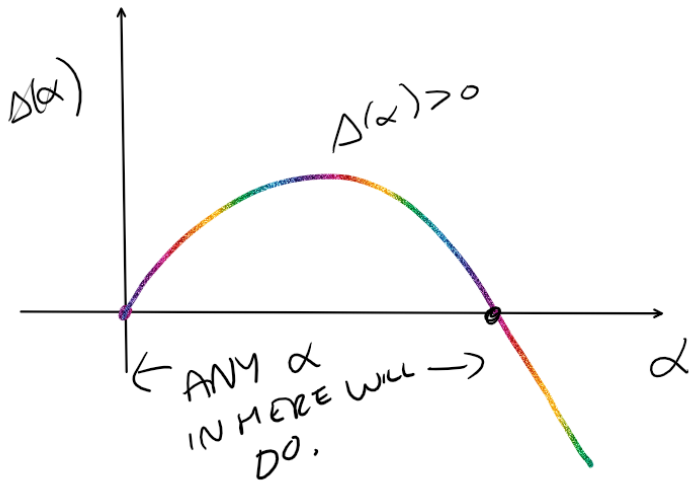


Figure 5.2: The graph of $\Delta(\alpha)$ showing the existence of the nonzero root $\Delta(\alpha_0) = 0$.

So by continuity of ϕ, there exists $\alpha_0 > 0$ such that $\Delta(\alpha_0) = 0$, and $\Delta(\alpha) \geq 0$ for $0 < \alpha < \alpha_0$. Thus, the first SWC is satisfied for any $0 < \alpha < \alpha_0$.

For the second SWC, we start with α_0 from the previous calculation, where

$$\Delta(\alpha_0) = 0 \implies \phi(\alpha_0) = \phi(0) + \alpha_0 c_1 \phi'(0).$$

Recall Taylor's Theorem,

$$\phi(\alpha_0) = \phi(0) + \phi'(\beta)\alpha_0, \qquad \beta \in (0, \alpha_0),$$

Combine these results:

$$c_1\phi'(0) = \phi'(\beta), \qquad \beta \in (0, \alpha_0),$$

These are negative slops, we work with:

$$|\phi'(\beta)| = c_1|\phi'(0)| \qquad \beta \in (0, \alpha_0).$$

Take $c_2 > c_1$, hence

$$|\phi'(\beta)| = c_1|\phi'(0)| < c_2|\phi'(0)| \qquad \beta \in (0, \alpha_0).$$

Thus, $0 < \beta < \alpha_0$ satisfies the two SWC. ∎

5.2 Wolfe Conditions

A slightly less stringent requirement for the step size is given by the Wolfe Conditions:

$$\phi(\alpha) \leq \phi(0) + c_1\phi'(0)\alpha, \tag{5.4a}$$

$$\phi'(\alpha) > c_2\phi'(0). \tag{5.4b}$$

The SWCs ensure that the magnitude of the slope is always controlled. The Wolfe Conditions ensure that the slope is always greater than a threshold value $c_2\phi'(0)$, but allow for slopes that may be too large and positive.

5.3 Backtracking Line Search

A quick and easy alternative implementation of the SWCs is the Backtracking Line Search algorithm. The idea here is to start with a step size that is too large, and reduce it so that the first SWC is satisfied. This actually makes the second SWC redundant.

Algorithm 4 Backtracking Line Search

Choose an initial guess for α_k, call it α. Fix $\rho \in (0, 1)$ and $c \in (0, 1)$.
while $f(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k) > f(\boldsymbol{x}_k) + c\alpha\boldsymbol{p}_k \cdot \nabla f(\boldsymbol{x}_k)$ **do**
  $\alpha \leftarrow \rho\alpha$.
end while

- Initial step length can be chosen as $\alpha_k = 1$ in Newton and Quasi-Newton methods.
- As there is no 'good' guess for the initial step length in SD, this method may not be appropriate there.

Hence, we look at a more involved implementation of the SWCs in the next section.

5.4 Practical Implementation of the SWCs

In the following code listings, we look at an implementation of the Line Search Algorithm in Matlab. Two initial guesses for α are chosen: $\alpha_0 = 0$, and $\alpha_1 \in (\alpha_0, \alpha_{max})$. The SWC are checked, if the

SWC conditions are violated, then α_1 is updated and the process is repeated. Specifically,

- If SW1 is violated, then we zoom in on an $\alpha_* \in (\alpha_0, \alpha_1)$ where SW1–SW2 are actually satisfied.
- If SW2 is violated, we actually increase the value of α_1 to a new value $\rho\alpha_0 + (1-\rho)\alpha_{max}$, where $0 < \rho < 1$.
- In the unusual situation where $\phi'(\alpha 1) > 0$, we would zoom in on an interval where SW1–2 are satisfied.

The idea is to keep iterating this process until SW1–2 are satisfied. The Matlab code for this is shown below.

```
function alpha_star = lsa(x,p,alpha_i)

% The following is an implementaiton of Algorithm 3.5, Nocedal and Wright,
% page 60.

% Note: This code is a simple modified version of a code by Davide Taviani,
% https://gist.github.com/Heliosmaster/1043132

% alpha_1 = a_i
% a0 = a_{i-1}

% Initialization of default parameters
% Here, c1 and c2 are the parameters s recommended by Nocedal and Wright
% (p. 62)

c1 = 1e-4;
c2 = 0.9;
maxit = 100;

alpha_0 = 0;                    % 0th steplength is 0
alpha_max = 10*alpha_i;         % Max serch interval is 10*(initial guess)

[f0,g0] = fun(x);               % Function values at \alpha=0.
df0=dot(p,g0);

i=1;

while 1
    fold = fun(x+alpha_0*p);     % Function with previous step-length
    [f,g] = fun(x+alpha_i*p);    % Function with current step-length
    df=dot(g,p);

    if( (f > f0+c1*alpha_i*df0) || ((i>1) && (f > fold) ))  % Check for SW1: A sufficent decrease in f or a comparison
         between f and fold)
        alpha_star = zoom_w(x,p,alpha_0,alpha_i,c1,c2);     % Modify alpha: a suitable value is somewhere in [alpha_0,
             alpha_1]
        return;
    end

    if(abs(df) <= -c2*df0)          % Check for SW2 (Curvature condition)
        alpha_star = alpha_i;       % Current step-length satisfies SW2
        return;
    end

    if(df >= 0)
        alpha_star = zoom_w(x,p,alpha_i,alpha_0,c1,c2); % Find a suitable step-length in [alpha_1,alpha_0]
        return;
    end

    if(i == maxit)              % Break Clause
        disp('Maximum number of iteration for Line Search reached');
```

```
50          alpha_star = alpha_i;
51          return;
52      end
53
54      % Update for next loop
55      i=i+1;
56      alpha_0 = alpha_i;
57
58      % Here, I am updating alpha_{i+1} using linear interplation based on
59      % rho, rho = 0.8;
60
61      rho=0.8;
62      alpha_i = rho*alpha_0+(1-rho)*alpha_max;
63
64      % alpha_1 = min(alpha_max, alpha_1*3);
65
66  end
67
68  end
```

The idea of the Zoom function here is as follows:

- Obtain an interval bounded by α_{lo} and α_{hi} such that there is an α in the interval that satisfies the SWCs.
- Of all the step lengths generated so far which satisfy SW1, α_{lo} is the one that gives the lowest value of $\phi(\alpha)$;
- α_{hi} is chosen such that $\phi'(\alpha_{\text{lo}})(\alpha_{\text{hi}} - \alpha_{\text{lo}}) < 0$.

```
function alpha_star = zoom_w(x,p,alpha_lo,alpha_hi,c1,c2)

% The following is an implementaiton of Algorithm 3.6, Nocedal and Wright,
% page 61.

% Note: This code is a simple modified version of a code by Davide Taviani,
% https://gist.github.com/Heliosmaster/1043132

maxit = 20;

[f0,g0] = fun(x);
df0=dot(p,g0);

j = 0;
while 1
    % I use bracketing and bisection to estimate the best value of alpha,
    % meaning the trial step-length is the middle point of [alpha_lo,alpha_hi]

    alpha_j = (alpha_lo+alpha_hi)/2;

    [f,g] = fun(x+alpha_j*p);
    df=dot(p,g);

    [f_lo] = fun(x+alpha_lo*p);

    if ( (f > f0 + c1*alpha_j*df0) || (f >= f_lo) )   % Test for sufficient decrease or comparison with alpha_lo.
        alpha_hi = alpha_j;                           % Narrow the interval between [alpha_lo,alpha_hi]
    else
        if abs(df) <= -c2*df0               % Curvature condition (SW2) satisfied
            alpha_star = alpha_j;
            return;
        end
        if df*(alpha_hi-alpha_lo) >= 0
            alpha_hi = alpha_lo;
        end
        alpha_lo = alpha_j;                 % The interval is now [alpha,alpha_lo]
    end

    if j==maxit
        alpha_star = alpha_j;               % Break Clause
        return;
    end

    j = j+1;
end

end
```

Chapter 6

Linesearch Methods – Convergence analysis

Overview

6.1 Convergence of Line Search Methods

To study the convergen of Line Search Methods, the specific angle between the descent direction $\boldsymbol{p}_k$ and the steepest-descent direction $-\nabla f(\boldsymbol{x}_k)/|\nabla f(\boldsymbol{x}_k)|$ is of interest. This can be obtained from the dot product:

$$\cos\theta_k = -\frac{\langle \boldsymbol{p}_k, \nabla f\rangle}{\|\boldsymbol{p}_k\|_2 \|\nabla f(\boldsymbol{x}_k)\|_2}.$$

We now prove the following theorem:

Theorem 6.1 (Zoutendijk Condition) *Consider an iterative line search method of the form $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$, where $\boldsymbol{p}_k$ is a descent direction, $\nabla f(\boldsymbol{x}) \cdot \boldsymbol{p}_k < 0$. Suppose that α_k satisfies the SWCs. Suppose also the following:*

1. *f is bounded below in $\mathbb{R}^n$;*
2. *f is continuously differentiable in an open set $\mathcal{N}$ containing the level sets*

$$\mathcal{L} = \{\boldsymbol{x} | f(\boldsymbol{x}) \leq f(\boldsymbol{x}_0)\},$$

   where $\boldsymbol{x}_0$ is the starting-value of the iterative method.
3. *∇f is Lipschitz continuous in $\mathcal{N}$, that is, there exists a constant $L > 0$ such that:*

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \leq L\|\boldsymbol{x} - \boldsymbol{y}\|_2, \qquad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathcal{N}.$$

Then:

$$\sum_{k\geq 0} \cos^2\theta_k \|\nabla f(\boldsymbol{x}_k)\|_2^2 < \infty.$$

Proof: We begin by recalling SW2. We have two cases to look at: $\nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)\cdot \boldsymbol{p}_k > 0$ and $\nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)\cdot \boldsymbol{p}_k \neq 0$. For brevity, we identify $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$ and write $\nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) = \nabla f(\boldsymbol{x}_{k+1}) = \nabla f_{k+1}$, etc. In Case 1, $\nabla f_{k+1}\cdot \boldsymbol{p}_k > 0$, and SW2 becomes:

$$\nabla f_{k+1}\cdot \boldsymbol{p}_k \leq -c_2 \nabla f_k \cdot \boldsymbol{p}_k,$$

since $|\nabla f_k \cdot \boldsymbol{p}_k| = -\nabla f_k \cdot \boldsymbol{p}_k$. Hence,

$$-\nabla f_{k+1}\cdot \boldsymbol{p}_k \leq \nabla f_{k+1}\cdot \boldsymbol{p}_k \leq -c_2 \nabla f_k \cdot \boldsymbol{p}_k,$$

and reversing the sign of the inequalities and multiplying by -1 gives:

$$\nabla f_{k+1}\cdot \boldsymbol{p}_k \geq c_2 \nabla f_k \cdot \boldsymbol{p}_k.$$

In Case 2, $\nabla f_{k+1}\cdot \boldsymbol{p}_k < 0$, and SW2 gives:

$$-\nabla f_{k+1}\cdot \boldsymbol{p}_k < -c_2 \nabla f_k \cdot \boldsymbol{p}_k.$$

Again reversing the sign of the inequalities and multiplying by -1 gives:

$$\nabla f_{k+1}\cdot \boldsymbol{p}_k > c_2 \nabla f_k \cdot \boldsymbol{p}_k.$$

So whichever case we are in, we have the key result:

$$\nabla f_{k+1}\cdot \boldsymbol{p}_k \geq c_2 \nabla f_k \cdot \boldsymbol{p}_k, \qquad \text{All Cases.}$$

Now, I can subtract $\nabla f_k \cdot \boldsymbol{p}_k$ from both sides to obtain:

$$(\nabla f_{k+1} - \nabla f_k)\cdot \boldsymbol{p}_k \geq (c_2 - 1)\nabla f_k \cdot \boldsymbol{p}_k, \qquad \text{All Cases.} \tag{6.1}$$

We also have the Lipschitz condition

$$\|\nabla f_{k+1} - \nabla f_k\|_2 \leq L\|\alpha_k \boldsymbol{p}_k\|_2,$$

hence

$$\|\nabla f_{k+1} - \nabla f_k\|_2 \|\boldsymbol{p}_k\| \leq L\alpha_k \|\boldsymbol{p}_k\|_2^2,$$

and using CS, this becomes:

$$|(\|\nabla f_{k+1} - \nabla f_k) \cdot \boldsymbol{p}_k| \leq \|\nabla f_{k+1} - \nabla f_k\|_2 \|\boldsymbol{p}_k\|_2 \leq L\alpha_k \|\boldsymbol{p}_k\|_2^2,$$

and more strongly,

$$(\nabla f_{k+1} - \nabla f_k) \cdot \boldsymbol{p}_k \leq L\alpha_k \|\boldsymbol{p}_k\|_2^2. \tag{6.2}$$

Combine inequalities (6.1) and (6.2) to obtain:

$$(c_2 - 1)\nabla f_k \boldsymbol{p}_k \leq L\alpha_k \|\boldsymbol{p}_k\|_2^2,$$

hence

$$\alpha_k \geq \frac{(c_2 - 1)\nabla f_k \cdot \boldsymbol{p}_k}{L\|\boldsymbol{p}_k\|_2^2},$$

or more transparently,

$$\alpha_k \geq \frac{(1 - c_2)|\nabla f_k \cdot \boldsymbol{p}_k|}{L\|\boldsymbol{p}_k\|_2^2}, \tag{6.3}$$

That is all I can do using SW2. So now go back to SW1:

$$f_{k+1} \leq f_k + \alpha_k c_1 \nabla f_k \cdot \boldsymbol{p}_k,$$

and sub in for α_k:

$$\begin{aligned} f_{k+1} &\leq f_k - c_1 \left(\frac{(1 - c_2)|\nabla f_k \cdot \boldsymbol{p}_k|}{L\|\boldsymbol{p}_k\|_2^2} \right) |\nabla f_k \cdot \boldsymbol{p}_k|, \\ &= f_k - \left[\frac{c_1(1 - c_2)}{L} \right] \frac{|\nabla f_k \cdot \boldsymbol{p}_k|^2}{\|\boldsymbol{p}_k\|_2^2}, \\ &= f_k - c \cos^2 \theta_k \|\nabla f_k\|_2^2. \end{aligned}$$

I sum over all k here to obtain:

$$f_{k+1} \leq f_0 - c \sum_{j=0}^{k} \cos^2 \theta_j \|\nabla f_j\|_2^2.$$

Since f is bounded below, I have $f_{k+1} \geq f_{min}$, hence $-f_{k+1} \leq -f_{min}$, hence

$$f_0 - f_{k+1} \leq f_0 - f_{min} = M,$$

where M is a constant (zero or positive), hence

$$c \sum_{j=0}^{k} \cos^2 \theta_j \|\nabla f_j\|_2^2 \leq M,$$

thus the sum is bounded. Taking $k \to \infty$, I have:

$$c \sum_{j=0}^{\infty} \cos^2 \theta_j \|\nabla f_j\|_2^2 \leq M,$$

which concludes the proof. ∎

Corollary 6.1 *If the conditions in Zoutendijk's Theorem are satisfied, then*

$$\cos^2 \theta_j \|\nabla f_j\|_2^2 \to 0 \text{ as } j \to \infty.$$

Proof: From Zoutendijk's Theorem, the series

$$\sum_{j=0}^{\infty} \cos^2 \theta_j \|\nabla f_j\|_2^2$$

is a convergent series, thus the general term goes to zero as $j \to \infty$:

$$\cos^2 \theta_j \|\nabla f_j\|_2^2 \to 0 \text{ as } j \to \infty. \quad \blacksquare$$

As an application of this Corollary, suppose that $\cos \theta_k$ can be kept away from zero in the 'tail' of the sequence of terms

$$\cos^2 \theta_0 \|\nabla f_0\|_2^2, \cos^2 \theta_1 \|\nabla f_1\|_2^2, \cdots, \cos^2 \theta_j \|\nabla f_j\|_2^2, \cdots$$

such that:

$$\cos \theta_k \geq \delta > 0, \qquad \text{for all } k \geq k_0,$$

Then,

$$\|\nabla f_j\|_2^2 \to 0 \text{ as } j \to \infty,$$

and in such a case, the iterative method converges.

6.1.1 Example

In the standard steepest-descent method, we have $\cos \theta_k = -1$ for all k, thus the SD method is guaranteed to converge, provided all of the conditions in Zoutendijk's Theorem are satisfied.

6.2 Application to Quasi-Newton methods

Zoutendijk's Theorem can also be applied to Quasi-Newton methods, where the descent direction is defined by:

$$B_k \boldsymbol{p}_k = -\nabla f_k,$$

provided the matrix B_k satisfies certain sensible conditions. This is made clear in the following Theorem:

Theorem 6.2 *Consider an iterative method where the descent direction is given by*

$$B_k \boldsymbol{p}_k = -\nabla f_k,$$

where B_k is a positive-definite matrix satisfying

$$\|B_k\|_2 \|B_k^{-1}\|_2 \leq M, \qquad M = \textit{Const. for all } k,$$

Then $\cos\theta_k \geq 1/M$.

A word first about $\kappa(B_k) := \|B_k\|_2 \|B_k^{-1}\|_2$: this is the **condition number** of the matrix B_k. In this context, $\|\cdot\|_2$ is the L^2 matrix norm:

$$\|B_k\|_2 = \sup_{\|\boldsymbol{u}\|_2 = 1} \|B_k \boldsymbol{u}\|_2.$$

If B_k is a positive-definite matrix, then it has all positive eigenvalues, and it can be readily shown that

$$\|B_k\|_2 = \max(\lambda_1, \cdots, \lambda_n) = \lambda_{max},$$

and also,

$$\|B_k^{-1}\|_2 = \max(\frac{1}{\lambda_1}, \cdots, \frac{1}{\lambda_n}) = \frac{1}{\lambda_{min}} > 0.,$$

Thus, for a positive-definite matrix such as B_k, the condition number can be written as

$$\kappa(B_k) = \|B_k\|_2 \|B_k^{-1}\|_2 = \frac{\lambda_{max}}{\lambda_{min}}.$$

We now return to the **proof of Theorem 6.2**. We have:

$$\cos\theta_k = \frac{\langle \nabla f_k, B_k^{-1} \nabla f_k \rangle}{\|\nabla f_k\|_2 \|B_k^{-1} \nabla f_k\|_2}.$$

As B_k is positive-definite, we can expand ∇f_k in terms of the eigenbasis of B_k:

$$\nabla f_k = \sum_i x_i \boldsymbol{u}_i, \qquad x_i = \langle \boldsymbol{u}_i, \nabla f_k \rangle, \qquad B_k \boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i,$$

hence

$$B_k^{-1} \nabla f_k = \sum_i \frac{1}{\lambda_i} x_i \boldsymbol{u}_i,$$

Thus,

$$\begin{aligned}
\cos\theta_k &= \frac{\langle \nabla f_k, B_k^{-1} \nabla f_k \rangle}{\|\nabla f_k\|_2 \|B_k^{-1} \nabla f_k\|_2}, \\
&= \frac{\sum_i \frac{1}{\lambda_i} x_i^2}{(\sum_i x_i^2)^{1/2} \left[\sum_i (1/\lambda_i)^2 x_i^2\right]^{1/2}}, \\
&\geq \frac{\frac{1}{\lambda_{max}} \sum_i x_i^2}{(\sum_i x_i^2)^{1/2} \left[\sum_i (1/\lambda_i)^2 x_i^2\right]^{1/2}}, \\
&\geq \frac{\frac{1}{\lambda_{max}} \sum_i x_i^2}{(\sum_i x_i^2)^{1/2} \left[(1/\lambda_{min})^2 \sum_i x_i^2\right]^{1/2}}, \\
&= \frac{\lambda_{min}}{\lambda_{max}}, \\
&= \frac{1}{\kappa(B_k)}, \\
&\geq 1/M.
\end{aligned}$$

Thus, $\cos\theta_k \geq 1/M$, as required. ∎

6.3 Convergence Rates – Steepest Descent

Often, it is important to know not only that a particular Line Search method converges but also, how fast it converges. For the standard Steepest Descent Method, an exact result is known for the model quadratic problem:

$$f(\boldsymbol{x}) = c + \langle \boldsymbol{a}, \boldsymbol{x} \rangle + \tfrac{1}{2} \langle \boldsymbol{x}, B\boldsymbol{x} \rangle. \tag{6.4}$$

and the result shows that the convergence is quite poor (linear). We go through the calculation here.

We start by computing the steepest-descent direction associated with Equation (6.4),

$$\boldsymbol{p} = -\nabla f = -B\boldsymbol{x},$$

and we assume the update step is given as:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) := \boldsymbol{x}_k - \alpha_k \nabla f_k. \tag{6.5}$$

To choose α_k, we solve the sub-problem:

$$\alpha_k = \arg\min_{\alpha > 0} f(\boldsymbol{x}_k - \alpha \nabla f_k),$$

for the quadratic cost function this has an exact solution:

$$\alpha_k = \frac{\langle \nabla f_k, \nabla f_k \rangle}{\langle \nabla f_k, B \nabla f_k \rangle}. \tag{6.6}$$

Consider again the update step (6.5), subtract $\boldsymbol{x}_*$ from both sides to obtain:

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_* = \boldsymbol{x}_k - \boldsymbol{x}_* - \alpha_k \nabla f_k.$$

Multiply both sides by B to obtain:

$$B(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*) = B(\boldsymbol{x}_k - \boldsymbol{x}_*) - \alpha_k B \nabla f_k.$$

Now take the inner product of both sides with $\boldsymbol{x}_{k+1} - \boldsymbol{x}_*$:

$$\begin{aligned} \langle \boldsymbol{x}_{k+1} - \boldsymbol{x}_*, B(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*) \rangle &= \langle \boldsymbol{x}_{k+1} - \boldsymbol{x}_*, B(\boldsymbol{x}_k - \boldsymbol{x}_*) - \alpha_k B \nabla f_k \rangle, \\ &= \langle \boldsymbol{x}_k - \boldsymbol{x}_* - \alpha_k \nabla f_k, B(\boldsymbol{x}_k - \boldsymbol{x}_*) - \alpha_k B \nabla f_k \rangle, \end{aligned}$$

Expand out to obtain:

$$\langle \boldsymbol{x}_{k+1} - \boldsymbol{x}_*, B(\boldsymbol{x}_{k+1} - \boldsymbol{x}_*) \rangle = \langle \boldsymbol{x}_k - \boldsymbol{x}_*, B(\boldsymbol{x}_k - \boldsymbol{x}_*) \rangle - 2\alpha_k \langle \boldsymbol{x}_k - \boldsymbol{x}_*, B \nabla f_k \rangle + \alpha_k^2 \langle \nabla f_k, B \nabla f_k \rangle. \tag{6.7}$$

We identify a weighted norm:

$$\|\boldsymbol{v}\|_B^2 = \langle \boldsymbol{v}, B\boldsymbol{v} \rangle, \qquad \text{for all } \boldsymbol{v} \in \mathbb{R}^n,$$

with $\|\boldsymbol{v}\|_B \geq 0$ and $\|\boldsymbol{v}\|_B = 0$ if and only if $\boldsymbol{v} = 0$. Hence, Equation (6.7) can be re-written as:

$$\underbrace{\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_B^2 - \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_B^2}_{=\Delta} = 2\alpha_k \langle \boldsymbol{x}_k - \boldsymbol{x}_*, B \nabla f_k \rangle - \alpha_k^2 \langle \nabla f_k, B \nabla f_k \rangle.$$

We now use:

$$\begin{aligned}\nabla f_k &= \boldsymbol{a} + B\boldsymbol{x}_k,\\ &= -B\boldsymbol{x}_* + B\boldsymbol{x}_k,\\ &= B(\boldsymbol{x}_k - \boldsymbol{x}_*).\end{aligned}$$

Hence:

$$\begin{aligned}\Delta &= 2\alpha_k\langle \boldsymbol{x}_k - \boldsymbol{x}_*, B\nabla f_k\rangle - \alpha_k^2\langle \nabla f_k, B\nabla f_k\rangle,\\ &= 2\alpha_k\langle B(\boldsymbol{x}_k - \boldsymbol{x}_*), \nabla f_k\rangle - \alpha_k^2\langle \nabla f_k, B\nabla f_k\rangle,\\ &= 2\alpha_k\langle \nabla f_k, \nabla f_k\rangle - \alpha_k^2\langle \nabla f_k, B\nabla f_k\rangle.\end{aligned}$$

Now fill in for α_k:

$$\Delta = 2\frac{\langle \nabla f_k, \nabla f_k\rangle}{\langle \nabla f_k, B\nabla f_k\rangle} \times \langle \nabla f_k, \nabla f_k\rangle - \frac{\langle \nabla f_k, \nabla f_k\rangle}{\langle \nabla f_k, B\nabla f_k\rangle} \times \frac{\langle \nabla f_k, \nabla f_k\rangle}{\langle \nabla f_k, B\nabla f_k\rangle} \times \langle \nabla f_k, B\nabla f_k\rangle$$

Carry out the cancellations to obtain:

$$\Delta = \frac{\langle \nabla f_k, \nabla f_k\rangle^2}{\langle \nabla f_k, B\nabla f_k\rangle}.$$

From the definition of Δ we now have:

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_B^2 = \|\boldsymbol{x}_k - \boldsymbol{x}_*\|_B^2 - \frac{\langle \nabla f_k, \nabla f_k\rangle^2}{\langle \nabla f_k, B\nabla f_k\rangle}. \tag{6.8}$$

But also, $\boldsymbol{x}_k - \boldsymbol{x}_* = B^{-1}\nabla f_k$, hence

$$\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_B^2 = \langle \nabla f_k, B^{-1}\nabla f_k\rangle.$$

So now we have:

$$\begin{aligned}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_B^2 &= \|\boldsymbol{x}_k - \boldsymbol{x}_*\|_B^2 - \frac{\langle \nabla f_k, \nabla f_k\rangle^2}{\langle \nabla f_k, B\nabla f_k\rangle},\\ &= \|\boldsymbol{x}_k - \boldsymbol{x}_*\|_B^2 - \frac{\langle \nabla f_k, \nabla f_k\rangle^2}{\langle \nabla f_k, B\nabla f_k\rangle}\frac{\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_B^2}{\langle \nabla f_k, B^{-1}\nabla f_k\rangle},\\ &= \left(1 - \frac{\langle \nabla f_k, \nabla f_k\rangle^2}{\langle \nabla f_k, B\nabla f_k\rangle\langle \nabla f_k, B^{-1}\nabla f_k\rangle}\right)\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_B^2.\end{aligned}$$

We notice the combination

$$\begin{aligned}\frac{\langle \nabla f_k, B\nabla f_k\rangle\langle \nabla f_k, B^{-1}\nabla f_k\rangle}{\langle \nabla f_k, \nabla f_k\rangle^2} &\leq \|B\|_2\|B^{-1}\|_2,\\ &= \kappa(B),\\ &= \frac{\lambda_{max}}{\lambda_{min}},\end{aligned}$$

where $\kappa(B)$ is the condition number of the matrix. Thus,

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_B^2 \leq \left(1 - \frac{1}{\kappa(B)}\right)\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_B^2,$$

hence

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_B \leq \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max}}\right)^{1/2}\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_B, \tag{6.9}$$

which establishes a **linear rate of convergence** of for the SD method.

6.3.1 A Warning about Scaling

Equation (6.9) reveals what happens to the performance of the SD method in the case of ill-conditioned problems where $\kappa(B)$ is large – then the $\lambda_{max} \gg \lambda_{min}$, and the prefactor in Equation (6.9) is very close to one, in which case:

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_B \lesssim \|\boldsymbol{x}_k - \boldsymbol{x}_*\|_B, \tag{6.10}$$

in which case the convergence is very poor. Geometrically, the level sets of such an ill-conditioned cost function look like very elongated ellipses (Figure 6.1), meaning that the steepest-descent method does not do a very good job at taking us towards the minimum.

6.3.2 A Warning about the SD method

Even for well-conditioned problems, the linear convergence of the SD method is regarded as quite poor. Geometrically, the level sets of f look like ellipses, the SD method takes us in steps towards the minimum, each step is orthogonal to a level set, meaning the path to the minimum is a 'zig-zag', which slows convergence (Figure 6.2). For this reason, Newton methods and Quasi-Newton methods are preferable, because they exhibit quadratic convergence to the minimum.

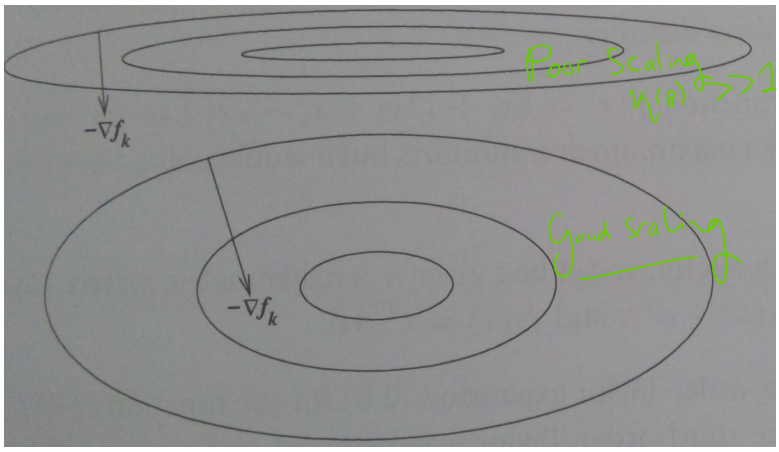


Figure 6.1: Example of a poorly-scaled cost function (top) and a well-scaled cost function (bottom), taken from Nocedal and Wright.

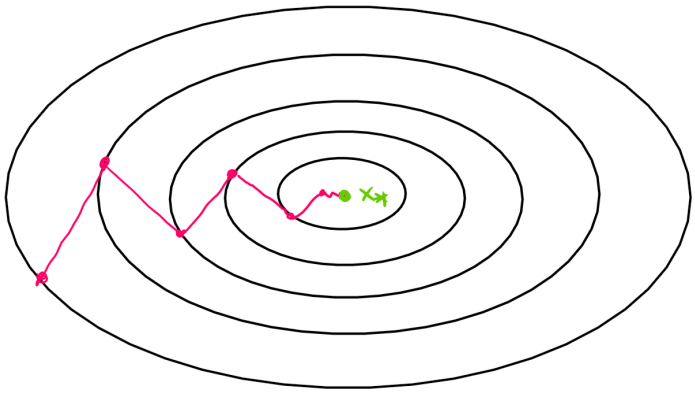


Figure 6.2: The 'zig-zag' SD path is responsible for the linear convergence rate of the SD method.

6.4 Convergence Rates – Newton

In Chapter 3 we sketched out the idea behind the quadratic convergence property for the Newton method. More specifically, for the update method:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{p}_k^N, \qquad B(\boldsymbol{x}_k)\boldsymbol{p}_k^N = -\nabla f(\boldsymbol{x}_k), \qquad [B(\boldsymbol{x}_k)]_{ij} = \left(\frac{\partial^2 f}{\partial x_i \partial x_j}\right)_{\boldsymbol{x}_k}$$

we expect (under certain restrictions on the Hessian B) that the rate of convergence of the sequence $\boldsymbol{x}_k \to \boldsymbol{x}_*$ should be quadratic, in the sense that:

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_2 \leq C\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2^2,$$

where C is a positive constant: it can be problem-specific but is independent of k.

Theorem 6.3 *Suppose that f is twice differentiable and that the Hessian $B(\boldsymbol{x})$ is Lipschitz continuous in a neighborhood of a solution $\boldsymbol{x}_*$ at which the sufficient conditions (Theorem 2.7) are satisfied. Suppose that the starting point $\boldsymbol{x}_0$ is sufficiently close to $\boldsymbol{x}_*$, and consider the iteration $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{p}_k^N$. Then,*

1. *The sequence of iterates $\{\boldsymbol{x}_k\}_{k=0}^{\infty}$ converges to $\boldsymbol{x}_*$;*
2. *The rate of convergence of the sequence is quadratic.*

Proof: We have $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - B^{-1}\nabla f_k$, where the dependence of B^{-1} and B on $\boldsymbol{x}_k$ is assumed. Subtract $\boldsymbol{x}_*$ from both sides, and use $\nabla f(\boldsymbol{x}_*) = 0$:

$$\begin{aligned} \boldsymbol{x}_{k+1} - \boldsymbol{x}_* &= \boldsymbol{x}_k - \boldsymbol{x}_* - B^{-1}\nabla f(\boldsymbol{x}_k), \\ &= \boldsymbol{x}_k - \boldsymbol{x}_* - B^{-1}\left[\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)\right]. \end{aligned}$$

Hence,

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_* = B^{-1}\Big\{B(\boldsymbol{x}_k - \boldsymbol{x}_*) - \left[\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)\right]\Big\}. \tag{6.11}$$

We use the first-order version of Taylor's theorem for a continuously differentiable function – which is really just the Fundamental Theorem of Calculus:

$$g(b) = g(a) + \int_0^1 g'(a + t(b-a))\mathrm{d}t$$

We apply this result to a generic function $\phi(\boldsymbol{x})$, between the points $\boldsymbol{x}_k$ and $\boldsymbol{x}_*$:

$$\phi(\boldsymbol{x}_k) = \phi(\boldsymbol{x}_*) + \int_0^1 \phi'(\boldsymbol{x}_k + t(\boldsymbol{x}_k - \boldsymbol{x}_*))\mathrm{d}t$$

or

$$\phi(\boldsymbol{x}_k) = \phi(\boldsymbol{x}_*) + \int_0^1 (\boldsymbol{x}_k - \boldsymbol{x}_*) \cdot \nabla f(\boldsymbol{x}_k + t(\boldsymbol{x}_k - \boldsymbol{x}_*))\mathrm{d}t$$

Now set $\phi = \partial f/\partial x_j$:

$$\begin{aligned}\frac{\partial f}{\partial x_j}(\boldsymbol{x}_k) &= \frac{\partial f}{\partial x_j}(\boldsymbol{x}_*) + \int_0^1 (\boldsymbol{x}_k - \boldsymbol{x}_*)_i \frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{x}_k + t(\boldsymbol{x}_* - \boldsymbol{x}_k))\mathrm{d}t, \\ &= \frac{\partial f}{\partial x_j}(\boldsymbol{x}_*) + \int_0^1 B_{ij}(\boldsymbol{x}_k + t(\boldsymbol{x}_* - \boldsymbol{x}_k))(\boldsymbol{x}_k - \boldsymbol{x}_*)_i \mathrm{d}t\end{aligned}$$

Hence,

$$[\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)]_j = \int_0^1 B_{ij}(\boldsymbol{x}_k + t(\boldsymbol{x}_k - \boldsymbol{x}_*))(\boldsymbol{x}_k - \boldsymbol{x}_*)_i \mathrm{d}t.$$

Without index notation, this becomes:

$$\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*) = \int_0^1 B(\boldsymbol{x}_k + t(\boldsymbol{x}_* - \boldsymbol{x}_k))(\boldsymbol{x}_* - \boldsymbol{x}_k)\mathrm{d}t.$$

We substitute this result back into Equation (6.11) to obtain:

$$\begin{aligned}\boldsymbol{x}_{k+1} - \boldsymbol{x}_* &= B^{-1}(\boldsymbol{x}_k)\Big\{ B(\boldsymbol{x}_k)(\boldsymbol{x}_k - \boldsymbol{x}_*) - [\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_*)] \Big\}, \\ &= B^{-1}(\boldsymbol{x}_k)\Big\{ B(\boldsymbol{x}_k)(\boldsymbol{x}_k - \boldsymbol{x}_*) - \int_0^1 B(\boldsymbol{x}_k + t(\boldsymbol{x}_* - \boldsymbol{x}_k))(\boldsymbol{x}_* - \boldsymbol{x}_k)\mathrm{d}t \Big\}, \\ &= B^{-1}(\boldsymbol{x}_k)\Big\{ \int_0^1 [B(\boldsymbol{x}_k) - B(\boldsymbol{x}_k + t(\boldsymbol{x}_* - \boldsymbol{x}_k))]\,\mathrm{d}t \Big\}(\boldsymbol{x}_k - \boldsymbol{x}_*).\end{aligned}$$

Take norms on both sides to obtain:

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_2 \leq \|B^{-1}(\boldsymbol{x}_k)\|_2 \|\boldsymbol{x}_k - \boldsymbol{x}_*\| \int_0^1 \|B(\boldsymbol{x}_k) - B(\boldsymbol{x}_k + t(\boldsymbol{x}_* - \boldsymbol{x}_k))\|_2 \mathrm{d}t.$$

The Hessian is assumed to be Lipschitz continuous: hence, there exists a positive constant $L > 0$ such that, for all $\boldsymbol{x}$ and $\boldsymbol{y}$ in the region of interest, we have:

$$\|B(\boldsymbol{y}) - B(\boldsymbol{x})\|_2 \leq L\|\boldsymbol{y} - \boldsymbol{x}\|_2.$$

Hence,

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_2 \leq \|B^{-1}(\boldsymbol{x}_k)\|_2 \|\boldsymbol{x}_k - \boldsymbol{x}_*\| \int_0^1 t\Big\{ L\|(\boldsymbol{x}_* - \boldsymbol{x}_k))\|_2 \Big\}\mathrm{d}t,$$

and so:

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_2 \leq \tfrac{1}{2} L\|B^{-1}(\boldsymbol{x}_k)\|_2 \|(\boldsymbol{x}_* - \boldsymbol{x}_k))\|_2^2. \tag{6.12}$$

Furthermore, B and B^{-1} are continuous functions of $\boldsymbol{x}$, so, given any $\epsilon > 0$, there exist a $\delta > 0$

such that

$$\|B^{-1}(\boldsymbol{x}_k) - B^{-1}(\boldsymbol{x}_*)\|_2 < \epsilon \text{ whenever } \|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2 < \delta,$$

hence

$$\|B^{-1}(\boldsymbol{x}_k)\|_2 < \epsilon + \|B^{-1}(\boldsymbol{x}_*)\|_2.$$

Hence, for a good choice of ϵ,

$$\|B^{-1}(\boldsymbol{x}_k)\|_2 \leq 2\|B^{-1}(\boldsymbol{x}_*)\|_2 \text{ whenever } \|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2 < \delta.$$

So referring back to Equation (6.12), this becomes:

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_2 \leq L\|B^{-1}(\boldsymbol{x}_*)\|_2\|\boldsymbol{x}_* - \boldsymbol{x}_k\|_2^2 \text{ whenever } \|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2 < \delta.$$

Hence,

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_2 = C\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2^2 \text{ whenever } \|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2 < \delta. \tag{6.13}$$

Let us choose:

$$\|\boldsymbol{x}_0 - \boldsymbol{x}_*\|_2 < \delta, \qquad \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|_2 < \tfrac{1}{2C},$$

i.e.

$$\|\boldsymbol{x}_0 - \boldsymbol{x}_*\|_2 < \min\left(\delta, \tfrac{1}{2C}\right).$$

By Equation (6.13), we have:

$$\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_*\|_2}{\|\boldsymbol{x}_0 - \boldsymbol{x}_*\|_2} \leq C\|\boldsymbol{x}_0 - \boldsymbol{x}_*\| \leq \tfrac{1}{2}.$$

From this, it can be shown (exercises) that

$$\frac{\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2}{\|\boldsymbol{x}_0 - \boldsymbol{x}_*\|_2} \leq \frac{1}{2^{2^k} - 1},$$

hence

$$\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2 \to 0 \text{ as } k \to \infty. \tag{6.14}$$

Thus,

- Result (6.14) establishes convergence of the Newton Method, whenever $\|\boldsymbol{x}_0 - \boldsymbol{x}_*\|_2 < \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|_2 < \min[\delta, 1/(2C)]$.
- Results (6.13) and (6.14) establish that the convergence rate is quadratic. ∎

6.5 Convergence Rates – Quasi-Newton

Quasi-Newton methods also exhibit 'good' convergence – in this case, the descent method is computed as

$$\boldsymbol{p}_k = -B_k^{-1}\nabla f_k,$$

where now B_k is some **approximation** to the Hessian. Also in this case, the step length is not equal to one, but is set to α_k, where α_k is chosen to satisfy the SWCs. Here, by 'good' convergence, we mean 'more than linear but maybe not as good as quadratic', specifically,

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_2 \leq C\|\boldsymbol{x}_k - \boldsymbol{x}_*\|^{1+\epsilon}, \tag{6.15}$$

where ϵ is a positive constant, and C is also a positive constant that may be problem-specific but is independent of k. As with the proof of the quadratic convergence of the Newton method, Equation (6.15) assumes that the function f has various 'nice' properties and that the starting-value $\boldsymbol{x}_0$ is sufficiently close to the solution $\boldsymbol{x}_*$, where $\nabla f(\boldsymbol{x}_*) = 0$.

We won't go into this in detail – the approach to the proofs is the same as before, and we already have learned a lot about the pros and cons of SD, Newton, and Quasi-Newton. We will therefore just state the theorem for the Quasi-Newton Methods and then move on to new materials.

Theorem 6.4 *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable. Consider the iteration $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$, where $\boldsymbol{p}_k$ is a descent direction and α_k satisfies the Wolfe Conditions with $c_1 \leq 1/2$. If the sequence $\{\boldsymbol{x}_k\}_{k=0}^{\infty}$ converges to a point $\boldsymbol{x}_*$ such that $\nabla f(\boldsymbol{x}_*) = 0$ and $[B(\boldsymbol{x})]_{ij} = \partial^2 f(\boldsymbol{x})/\partial x_i \partial x_j$ is a positive definite matrix, and if the search direction satisfies:*

$$\lim_{k\to\infty} \frac{\|\nabla f_k + B\boldsymbol{p}_k\|_2}{\|\boldsymbol{p}_k\|_2} \to 0,$$

then:

1. *The step length $\alpha_k = 1$ is admissible for all k greater than a certain index k_0;*
2. *If $\alpha_k = 1$ for all $k > k_0$, then the sequence $\{\boldsymbol{x}_k\}_{k=0}^{\infty}$ converges to $\boldsymbol{x}_*$ superlinearly.*

Chapter 7

Trust-Region Methods

Overview

We introduce the general idea of Trust-Region Methods and we formulate the simplest possible such method – the so-called Cauchy-Point Method.

7.1 Introduction

We begin by recalling the idea behind line search methods – the idea there is to pick a 'nice' direction $\boldsymbol{p}_k$ at each iteration, and then to step to the next iteration via the update rule $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$. Then, the optimization problem at each iteration step is reduced to solving a 1D sub-problem

$$\alpha_k = \arg\min_{\alpha > 0} f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k).$$

Line search methods are good when the Hessian $B_{ij} = \partial^2 f / \partial x_i \partial x_j$ is positive-definite, in which case Newton or Quasi-Newton methods produce an iterative method with super-linear convergence to the optimal point $\|\nabla f(\boldsymbol{x}_*)\|_2 = 0$. Trust-region methods are another class of iterative method that can be used to solve optimization problems where the Hessian is positive-definite. However, the trust-region methods can be 'tweaked' in a very intuitive way to accommodate Hessians that are not positive-definite. Thus, the trust-region methods are slightly more robust.

In this chapter, we introduce the idea behind the trust-region method in the case of positive-definite Hessians and then outline the extension to non-sign-definite Hessians later on.

7.2 The Idea

The idea behind the trust-region method is that at each iteration, we locally approximate the cost function by a quadratic:

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_k + \boldsymbol{p}) \approx f_k + \langle \boldsymbol{g}, \boldsymbol{p} \rangle + \tfrac{1}{2}\langle \boldsymbol{p}, B\boldsymbol{p} \rangle = m_k(\boldsymbol{p}). \tag{7.1}$$

Now, it makes sense here that the 'coefficients' in this Taylor expansion should be $\boldsymbol{g} = \nabla f(\boldsymbol{x}_k)$ and $B_{ij} = \partial^2 f(\boldsymbol{x}_k)/\partial x_i \partial x_j$, although strictly speaking that is not necessary, the main thing is that $m_k(\boldsymbol{p})$ should be a good approximation to $f(\boldsymbol{x}_k + \boldsymbol{p})$ in Equation (7.1).

Equation (7.1) is a quadratic approximation to the cost function, as such, it should be valid in a small region centred on $\boldsymbol{x}_k$. We therefore introduce a **trust region** where we expect the quadratic approximation to be valid:

$$\|\boldsymbol{p}\|_2 \leq \Delta, \tag{7.2}$$

where Δ is the size of the trust region.

Once the size of the trust region has been established (we say how to do this in the next section), we can then solve the reduced problem:

$$\boldsymbol{p}_k = \arg \min_{\|\boldsymbol{p}\|_2 \leq \Delta} m_k(\boldsymbol{p}), \tag{7.3}$$

and hence, step from $\boldsymbol{x}_k$ to $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{p}_k$. As such, line search methods and trust-region methods are the same in spirit, the difference being that the sub-problem in the trust-region method is more complicated.

7.3 Size of Trust Region

There is a very simple algorithm for computing the size of the trust region: we compare

- The actual reduction in the cost function between iterations, $f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \boldsymbol{p}_k)$;
- The predicted reduction in the cost function between iterations, $m_k(0) - m_k(\boldsymbol{p}_k)$

Hence, we look at the ratio

$$\rho_k = \frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \boldsymbol{p}_k)}{m_k(0) - m_k(\boldsymbol{p}_k)} \tag{7.4}$$

and we will use the value of ρ_k as criterion for determining the size of the trust region, according to the following ideas:

- As $\boldsymbol{p}_k$ is supposed to be the minimum of $m_k(\boldsymbol{p})$ in a region containing $\boldsymbol{p} = 0$ (see Equation (7.3)), the denominator here is guaranteed to be positive for all iterations. Furthermore, if the numerator is negative, it means that our search direction $\boldsymbol{p}_k$ is not reducing the cost function, meaning that we should reject the search direction $\boldsymbol{p}_k$.

On the other hand,

- If ρ_k is positive and close to one, then there is good agreement between the predicted reduction in the cost function and the actual reduction in the cost function, in which case it is safe to expand the trust region at the next iteration.
- If ρ_k is positive but significantly smaller than one, then the trust region is still okay, but we leave it unchanged for the next iteration.
- If ρ_k is negative or much smaller than one, then we reduce the size of the trust region for the next iteration.

Algorithm 5 Determining Size of Trust Region

Choose a maximum size of the trust region, $\widehat{\Delta}$ and an initial guess for the size of the trust region, Δ_0. Also, choose a criterion $\eta \in [0, 1/4)$ for a descent direction to be accepted.

for $k = 0, 1, 2, \cdots$ **do**

Obtain $\boldsymbol{p}_k$ by (approximately) solving Equation (7.3).

Evaluate ρ_k from Equation (7.4).

if some condition is true **then**

$\Delta_{k+1} = (1/4)\Delta_k$

else

if $\rho_k > 3/4$ and $\|\boldsymbol{p}_k\|_2 = \Delta_k$ **then**

$\Delta_{k+1} = \min(2\Delta_k, \widehat{\Delta})$

else

$\Delta_{k+1} = \Delta_k$;

end if

end if

if $\rho_k > \eta$ **then**

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

else

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

end if

end for

7.4 The constrained minimization problem

The Trust-Region algorithm requires us to solve a **constrained** minimization problem at each iteration:

$$\boldsymbol{p}_* = \arg\min_{\boldsymbol{p}} \left[f_k + \langle \boldsymbol{g}_k, \boldsymbol{p} \rangle + \tfrac{1}{2} \langle \boldsymbol{p}, B_k \boldsymbol{p} \rangle \right], \text{ subject to } \|\boldsymbol{p}\|_2 \leq \Delta. \tag{7.5}$$

Here, there is a subscript k on $\boldsymbol{g}_k$ and B_k to indicate that these are the coefficients in the approximation to the cost function that are updated at each iteration. However, these indices are suppressed in what follows, as we seek to focus on the details of the minimization problem. Equation (7.5) can be solved using the method of Lagrange multipliers, we summarize the main result here. Notice that the subproblem has a solution for a general **symmetric** matrix B:

Theorem 7.1 *Suppose that B is a symmetric matrix. Then the vector $\boldsymbol{p}_*$ is a global solution of the trust-region problem*

$$\boldsymbol{p}_* = \arg\min_{\boldsymbol{p}} \left[f + \langle \boldsymbol{g}, \boldsymbol{p} \rangle + \tfrac{1}{2} \langle \boldsymbol{p}, B\boldsymbol{p} \rangle \right], \textit{ subject to } \|\boldsymbol{p}\| \leq \Delta, \tag{7.6}$$

if and only if $\boldsymbol{p}_*$ is feasible and there is a scalar $\lambda \geq 0$ such that the following conditions hold:

$$\begin{aligned} (B + \lambda \mathbb{I})\, \boldsymbol{p}_* &= -\boldsymbol{g}, \\ \lambda \left(\Delta - \|\boldsymbol{p}_*\|_2\right) &= 0, \\ (B + \lambda \mathbb{I}) &\quad \textit{is positive semi-definite.} \end{aligned}$$

Theorem 7.1 deals with exact solutions of the quadratic approximation. For now, we are only interested in approximate solutions (in analogy with the SWCs for Line Search Methods). Therefore, we can postpone our study of this theorem until a later chapter.

7.5 Approximate solution of the constrained minimization problem

The solution of Equation (7.6) is a vector $\boldsymbol{p}_*$ that depends parametrically on Δ, and we denote the solution by $\boldsymbol{p}_*(\Delta)$. Knowing the function $\boldsymbol{p}_*(\Delta)$ is like knowing the exact solution of $\alpha_k = \arg\min_{\alpha>0} f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k)$ in line search methods – it is not necessary to know this, a 'good' estimate for α_k (such as that given by the SWCs) is sufficient for convergence. The same idea carries over to the trust-region methods – knowing the solution $\boldsymbol{p}_*(\Delta)$ is too much, in practice a good estimate for $\boldsymbol{p}_*$ suffices.

7.6 Cauchy Point

The idea of the Cauchy Point method for solving the sub-problem is to look at the case when Δ is small. Then, the quadratic sub-problem can be approximated by

$$m_k(\boldsymbol{p}) \approx f_k + \langle \boldsymbol{g}, \boldsymbol{p} \rangle, \qquad \|\boldsymbol{p}\|_2 \leq \Delta$$

and the solution to the minimizing $m_k(\boldsymbol{p})$ is simply to take $\boldsymbol{p}$ along the steepest-descent path, $\boldsymbol{p} \propto -\boldsymbol{g}$. A first estimate of the solution is a vector which extends to the trust-region boundary:

$$\boldsymbol{p}_{\text{temp}} = -\frac{\Delta}{\|\boldsymbol{g}\|_2}\boldsymbol{g}.$$

We then refine this solution by taking:

$$\boldsymbol{p} = \tau \boldsymbol{p}_{\text{temp}},$$

where τ is a scalar to be determined. We determine τ by solving the full quadratic problem:

$$\tau = \arg\min_{\tau > 0} m_k(\boldsymbol{p}_{\text{temp}}\tau).$$

We further have the restriction $\|\tau \boldsymbol{p}_{\text{temp}}\|_2 \leq \Delta$, but this is the same as $|\tau| \leq 1$.

We now solve the minimization problem for τ. We look at $m_k(\tau \boldsymbol{p}_{\text{temp}})$, this becomes:

$$m_k(\tau \boldsymbol{p}_{\text{temp}}) = f_k - \frac{\tau\Delta\langle \boldsymbol{g}, \boldsymbol{g} \rangle}{\|\boldsymbol{g}\|_2} + \tfrac{1}{2}\tau^2\Delta^2\frac{\langle \boldsymbol{g}, B\boldsymbol{g} \rangle}{\|\boldsymbol{g}\|_2^2}.$$

As we we are dealing with the general trust-region method, we make no assumptions about whether B is a positive-definite matrix, so there are two cases to consider.

1. Case 1. We look at $\langle \boldsymbol{g}, B\boldsymbol{g} \rangle \leq 0$. Then, the minimum value of $m_k(\tau \boldsymbol{p}_{\text{temp}})$ occurs when $\tau = 1$, hence, the optimal vector is:

$$\boldsymbol{p} = 1 \times \boldsymbol{p}_{\text{temp}} = -\frac{\Delta}{\|\boldsymbol{g}\|_2}\boldsymbol{g}.$$

2. Case 2. We look at $\langle \boldsymbol{g}, B\boldsymbol{g} \rangle > 0$. Then, we must minimize the quadratic function

$$Q(\tau) = f_k - \tau\Delta\|\boldsymbol{g}\|_2 + \tfrac{1}{2}\tau^2\Delta^2\frac{\langle \boldsymbol{g}, B\boldsymbol{g} \rangle}{\|\boldsymbol{g}\|_2^2}.$$

   We find the optimal value of τ by setting $Q'(\tau) = 0$. Hence,

$$\tau = \frac{\|\boldsymbol{g}\|_2^3}{\Delta\langle \boldsymbol{g}, B\boldsymbol{g} \rangle}.$$

However, if this value exceeds one it can't be accepted, so the solution in Case 2 is:

$$\tau = \min\left(1, \frac{\|\boldsymbol{g}\|_2^3}{\Delta\langle \boldsymbol{g}, B\boldsymbol{g}\rangle}\right).$$

We summarize these result in the following theorem:

Theorem 7.2 *The Cauchy point is given by:*

$$\boldsymbol{p}_{Cauchy} = -\tau\left(\frac{\Delta}{\|\boldsymbol{g}\|_2}\boldsymbol{g}\right).$$

where,

$$\tau = \begin{cases} 1, & \text{if } \langle \boldsymbol{g}, B\boldsymbol{g}\rangle \leq 0, \\ \min\left(1, \frac{\|\boldsymbol{g}\|_2^3}{\Delta\langle \boldsymbol{g}, B\boldsymbol{g}\rangle}\right), & \text{otherwise.} \end{cases}$$

7.7 Worked Example

In this example, we use the Cauchy Point Algorithm to minimize the Rosenbrock function

$$f = 10(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

As this function is a simple function of two variables, the gradient and the Hessian can be computed analytically, these are supplied as part of the code. The global minimum can be found by inspection to be at $(x_1, x_2) = (1, 1)$, and our Cauchy algorithm finds this point after a couple of iterations. The sample code can be found in the listings below.

```
function [x] = trust_ros()

% tol: stopping criterion on the norm of gradient at current x:
tol = 1e-5;

% maxit: maximum number of iterations
maxit=10000;

% x0: initial guess for the TR method:
x0=rand(2,1);
x_k=x0;

% Trust-region parameters, see Algorithm 4.1 in Nocedal and Wright.
Delta_hat=0.01;
Delta_0=0.5*Delta_hat;
Delta_k=Delta_0;

eta=0.9*0.25;

for k=1:maxit

    % Calculation of the cost function.  Here, fun is the cost function,
    % this is defined in a separate Matlab routine and is called here.
    % The Hessian is known for this problem, that is why it is returned here.
```

```
    % The Hessian is stored in the array "B".

    [f_k,g,B] = fun(x_k);

    % I use the Cauchy Point method to estimate the descent direction p_k.
    % p_k=approx_solve_QP_cauchy(g,B,Delta_k);
    p_k=approx_solv_QP_dogleg(g,B,Delta_k);

    % I evaluuate the cost function at the corrected value, this will
    % determine if I need to reduce the trust region at the next step.
    f_corr=fun(x_k+p_k);

    % Now, I evaulate the quadratic approximation of f_k:
    m_0=fun_QP(f_k,g,B,0*p_k);
    m_k=fun_QP(f_k,g,B,p_k);

    % I compute rho_k for implementaiton of Algorithm 4.1 in Nocedal and
    % Wright:
    rho_k=(f_k-f_corr)/(m_0-m_k);

    if(rho_k<1/4)
        Delta_k_plus=(1/4)*Delta_k;
    else
        if( (rho_k>3/4) && ( norm(p_k)==Delta_k ))
            Delta_k_plus=min(2*Delta_k,Delta_hat);
        else
            Delta_k_plus=Delta_k;
        end
    end

    if( rho_k> eta)
        x_k_plus=x_k+p_k;
    else
        x_k_plus=x_k;
    end

    if(mod(k,10)==0)
        display(strcat('Iteration=',num2str(k),', |\nabla f|=',num2str(norm(g))))
    end

    if norm(g) < tol
        display(strcat('Convergence Reached in k=',num2str(k),' iterations, |\nabla f|=',num2str(norm(g))))
        break;
    end

    x_k=x_k_plus;
    Delta_k=Delta_k_plus;

end

x=x_k_plus;

end

function y=fun_QP(f,g,B,p)
    y=f+dot(p,g)+0.5*dot(p,B*p);
end

function p_cauchy=approx_solve_QP_cauchy(g,B,Delta)

    p_temp=-Delta*g/norm(g);

    if(dot(g,B*g)<=0)
        tau=1;
    else
        num=norm(g)^3;
        den=Delta*dot(g,B*g);
        tau=min(num/den,1);
    end
```

The output of the code is printed here: the code successfully finds the minimum of the cost function.

```
>> [x] = trust_ros()
Convergence Reached in k=440iterations, |\nabla f|=9.939e-06

x =

    1.0000
    1.0000
```

Figure 7.1: Output from the Trust-Region (Cauchy Point) algorithm

7.8 Drawback of Cauchy-Point Method

The general trust-region method is more complicated than the line search methods. The added complexity gives us scope to refine the search direction and the size of the search step at each iteration; it also allows us to extend line search methods to problems where the Hessian is not always positive-definite. However, the Cauchy Point is an overly simplistic implementation of the trust-region method: as we have set $\boldsymbol{p}_k \propto -\boldsymbol{g} = -\nabla f(\boldsymbol{x}_k)$ at each iteration, the Cauchy-Point Method is simply a steepest-descent method in disguise – albeit with a rather fancy method of choosing the step length.

Thus, the Cauchy-Point Method inherits all of the drawbacks of SD – including the linear rate of convergence, and the poor scaling. Therefore, in the next chapter we look at a more sophisticated Trust-Region method that overcomes some of these drawbacks.

Chapter 8

Dog-Leg Method

Overview

In the last chapter we introduced the general idea of Trust-Region Methods and we formulated the simplest possible such method – the so-called Cauchy-Point Method. The Cauchy-Point Method suffers from a major drawback in that it is essentially a dressed-up version of the Steepest Descent Method. It therefore possesses linear convergence and poor scaling. Therefore, in this chapter we look at more complex Trust-Region Methods that overcome some of these shortcomings. We look first at quadratic approximations to the cost function where the B-matrix is always positive definite. This leads to the so-called *Dogleg Method*. Then, at the end of the chapter we look at an implementation of the trust-region method where this assumption can be lifted.

8.1 The idea behind the Dogleg Method

Recall, in the Trust Region method, we are technically required to solve the quadratic approximation at each iteration. This is a constrained optimization problem ('the constrained sub-problem'):

$$\boldsymbol{p}_* = \arg\min_{\boldsymbol{p}} \left[f + \langle \boldsymbol{g}, \boldsymbol{p} \rangle + \tfrac{1}{2} \langle \boldsymbol{p}, B\boldsymbol{p} \rangle \right], \text{ subject to } \|\boldsymbol{p}\|_2 \leq \Delta. \tag{8.1}$$

The solution can be found exactly, and it is parametric function of the trust-region size, hence the solution is written as $\boldsymbol{p}_*(\Delta)$.

We specialize here to the case where B is **positive definite**. Imagine for a minute that there is no constraint. Then, the optimization problem (8.1) has an obvious (unique) solution:

$$\boldsymbol{p}_* = -B^{-1}\boldsymbol{g}. \tag{8.2}$$

Thus, the idea of the Dog-Leg method is to accept $\boldsymbol{p}_*$ as a solution to the constrained sub-problem so long as $\|B^{-1}\boldsymbol{g}\|_2 \leq \Delta$. Indeed, in this case, Equation (8.2) is an exact solution of Equation (8.1).

Notation: We can use the notation $\boldsymbol{p}_{\text{Newton}} = -B^{-1}\boldsymbol{g}$ in Equation (8.2) as this is just the Newton step from our earlier study of line search methods.

On the other hand, when $\|B^{-1}\boldsymbol{g}\|_2 > \Delta$, then Equation (8.2) is no longer a solution of the constrained sub-problem, and we instead introduce an **approximate** solution.

The first approximate solution we look for involves

$$\boldsymbol{p}_{\text{SD}} = -\frac{\langle \boldsymbol{g}, \boldsymbol{g} \rangle}{\langle \boldsymbol{g}, B\boldsymbol{g} \rangle}\boldsymbol{g} \tag{8.3}$$

which is the optimal descent direction in the Steepest-Descent Method (quadratic model problem). If $\boldsymbol{p}_{\text{SD}}$ lies outside the trust region, we rescale and get

$$\boldsymbol{p}_* \approx -(\Delta/\|\boldsymbol{g}\|_2)\boldsymbol{g},$$

as an approximate solution of the constrained sub-problem.

A third option is when $\boldsymbol{p}_{\text{Newton}}$ is outside the trust region but $\boldsymbol{p}_{\text{SD}}$ is inside the trust region. Then we may construct a linear combination,

$$\boldsymbol{p}_* \approx \boldsymbol{p}_{\text{SD}} + \alpha\left(\boldsymbol{p}_{\text{Newton}} - \boldsymbol{p}_{\text{SD}}\right), \qquad \alpha \in (0,1). \tag{8.4}$$

We then fix α such that $\boldsymbol{p}_*$ lies on the trust-region boundary:

$$\|\boldsymbol{p}_*\|_2^2 = \|\boldsymbol{p}_{\text{SD}} + \alpha\left(\boldsymbol{p}_{\text{Newton}} - \boldsymbol{p}_{\text{SD}}\right)\|_2^2 = \Delta^2. \tag{8.5}$$

Summarizing, we have the following three options:

- Case 1. If $\|\boldsymbol{p}_{\text{Newton}}\|_2 \leq \Delta$, then we accept $\boldsymbol{p}_* = \boldsymbol{p}_{\text{Newton}}$ as the solution of the constrained sub-problem.
- Otherwise:
  - Case 2.1: If $\|\boldsymbol{p}_{\text{SD}}\|_2 > \Delta$, then we accept $\boldsymbol{p}_* \approx -(\Delta/\|\boldsymbol{g}\|_2)\boldsymbol{g}$ as an approximate solution of the constrained sub-problem.
  - Otherwise:
    - Case 2.2: We accept Equation (8.4) as an approximate solution of the constrained sub-problem.

8.2 Analysis of Dogleg Method

To analyse whether the Dogleg Method works, we need to establish that a scalar value of α can be found such that Equation (8.5) can be solved. As such, we expand out the terms in Equation (8.5) and check that the conditions are fulfilled for the equation to have a real positive solution in α.

We expand out Equation (8.5) to get:

$$\alpha^2\|\boldsymbol{p}_{\text{Newton}} - \boldsymbol{p}_{\text{SD}}\|_2^2 + 2\alpha\langle\boldsymbol{p}_{\text{SD}}, \boldsymbol{p}_{\text{Newton}} - \boldsymbol{p}_{\text{SD}}\rangle + \|\boldsymbol{p}_{\text{SD}}\|_2^2 - \Delta^2 = 0. \tag{8.6}$$

Notice that this is a quadratic equation in α:

$$a\alpha^2 + 2b\alpha + c = 0.$$

The solution is:

$$\alpha = \frac{-b \pm \sqrt{b^2 - ac}}{a}.$$

Therefore, we need to determine first if real roots exist.

We look at $b^2 - ac$. However, as $c = \|\boldsymbol{p}_{\text{SD}}\|_2^2 - \Delta^2$, we have $c < 0$ (Case 2.2), hence $b^2 - ac = b^2 + a|c| \geq 0$ and hence, real roots exist. Trial-and-error then suggests to take the positive branch of the solution.

Observe finally that if we have

$$h(\alpha) = \|\boldsymbol{p}_{\text{SD}} + \alpha\left(\boldsymbol{p}_{\text{Newton}} - \boldsymbol{p}_{\text{SD}}\right)\|_2^2,$$

then $h(0) = \|\boldsymbol{p}_{\text{SD}}\|_2^2$ (less than Δ^2, in Case 2.2), and $h(1) = \|\boldsymbol{p}_{\text{Newton}}\|_2^2$ (greater than Δ^2, in Case 2.2). Thus, the value of α such that $h(\alpha) = \Delta^2$ must be between 0 and 1, as a graphical argument will very quickly show.

8.3 Implementation of the Dogleg Method

A sample .m code do this is shown in the listings.

```
function p_dogleg=approx_solv_QP_dogleg(g,B,Delta)

    % Check if B is positive definite.
    [~,flag] = chol(B);

    if(flag==0)

        % If flag==0 then the matrix is symmetric positive definite,
        % and I can do dogleg:
```

```
10
11          % Newton step
12          p_newton=-B\g;
13
14          % SD step, with stepsize obtained from solving the quadratic model
15          % problem (QMP):
16          p_SD=-(dot(g,g)/(dot(g,B*g)))*g;
17
18          if( norm(p_newton)<Delta)
19              % Here, if the Newton step is fully inside the trust-region,
20              % then I accept the Newton step as the solution of the
21              % constrained sub-problem.
22
23              p_dogleg=p_newton;
24
25          elseif( norm(p_SD)>=Delta)
26              % If this is not the case, then I have to do something else.
27              % The first thing I check is if the SD step is fully outside
28              % the trust region.  Then I have to accept as an approximate
29              % solution a scaled version of the SD step, on the trust-region
30              % boundary.
31
32              p_dogleg=-Delta*g/norm(g);
33          else
34              % Otherwise, the Newton step is outside the trust region and
35              % the SD step is inside the trust-region boundary, so I
36              % construct a linear combination of the two, on the
37              % trust-region boundary.
38
39              aa=dot(p_newton-p_SD,p_newton-p_SD);
40              bb=dot(p_SD,p_newton-p_SD);
41              cc= dot(p_SD,p_SD)-Delta*Delta ;
42
43              % I need to take the positive sign here.
44              alpha=(-bb+sqrt(bb*bb-aa*cc) )/aa
45
46              p_dogleg=p_SD+alpha*(p_newton-p_SD);
47          end
48      else
49
50          % Here, if the B-matrix is not symmetric positive-definite I have
51          % to revert to the Cauchy-point method.
52
53          p_temp=-Delta*g/norm(g);
54
55          if(dot(g,B*g)<=0)
56              tau=1;
57          else
58              num=norm(g)^3;
59              den=Delta*dot(g,B*g);
60              tau=min(num/den,1);
61          end
62
63          p_dogleg=tau*p_temp;
64      end
65
66  end
```

8.4 Discussion

8.4.1 The reason behind the funny name

The reason for the funny name for the Dogleg method is as follows. Consider the exact solution of the quadratic approximation, which we label by $\boldsymbol{p}_*(\Delta)$, to indicate the parametric dependence on

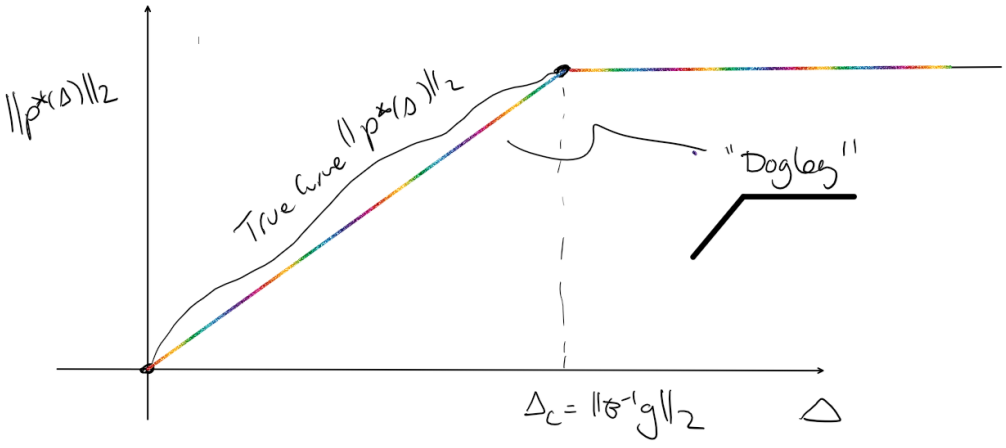


Figure 8.1: Plot showing the justification of the name of the Dogleg method

Δ. This solution can be regarded as a curve in parameter space. The Dogleg method really seeks to approximate this curve by two parts:

$$\boldsymbol{p}_* \approx \begin{cases} -B^{-1}\boldsymbol{g}, & \|B^{-1}\boldsymbol{g}\|_2 \leq \Delta, \\ -\tau \frac{\Delta}{\|\boldsymbol{g}\|_2}\boldsymbol{g}, & \text{otherwise.} \end{cases}$$

By plotting even the length of this piecewise curve as a function of Δ and by comparing it to the length of the true curve $\|\boldsymbol{p}_*(\Delta)\|_2$, a highly imaginative individual may think that the piecewise approximation looks like the leg of a dog (Figure 8.1).

8.4.2 Convergence of Trust-Region Methods

We have already addressed the linear convergence of the Cauchy-Point method, as this is essentially the SD method in disguise. We may expect that the Dogleg method is superior, as for sufficiently large trust regions, it reverts to the Newton method. Furthermore, as we get closer and closer to the solution $\nabla f(\boldsymbol{x}_*) = 0$, the quadratic approximation becomes closer and closer to the true cost function, leading us to expect that the Dogleg method may become closer and closer to the Newton method in fact as well as in spirit – and hence, to exhibit Newton-like superlinear convergence. This indeed turns out to be the case, and the Dogleg method can be shown to possess such super-linear convergence, once the cost function has 'nice' properties. This is discussed in detail in Nocedal and Wright (Section 4.4). However, as we have already looked at convergence proofs in a lot of detail, and are confident in the general techniques involved in such proofs, the reader is referred to Nocedal and Wright for more details.

8.5 B-matrices that are not positive-definite

We look now at the case when the B-matrix in the quadratic approximation $m(\boldsymbol{p}) = f + \langle \boldsymbol{g}, \boldsymbol{p} \rangle + (1/2)\langle \boldsymbol{g}, B\boldsymbol{g} \rangle$ is no longer positive definite. For instance, if B is the Hessian, then B is at least positive-semi-definite at the optimal point where $\nabla f(\boldsymbol{x}_*) = 0$, however, the Hessian at other points may be indefinite (e.g. 'saddle points'). In this case, so-called two-dimensional subspace minimization can be used to find a minimum (or at least, an approximation to the minimum) of $m(\boldsymbol{p})$ at each iteration. The different possibilities for the two-dimensional subspace minimization are enumerated here.

8.5.1 When B is positive-definite

Then, instead of solving the full sub-problem where we minimize $m(\boldsymbol{p})$ over all $\boldsymbol{p}$ subject to $\|\boldsymbol{p}\|_2 \leq \Delta$, we perform a two-dimensional subspace minimization:

$$\boldsymbol{p}_* = \arg\min m(\boldsymbol{p}), \qquad \boldsymbol{p} \in \mathrm{Span}(\boldsymbol{g}, B^{-1}\boldsymbol{g}), \qquad \|\boldsymbol{p}\|_2 \leq \Delta. \tag{8.7}$$

This is a two-dimensional minimization problem where we minimize over all α and β:

$$\min_{\alpha,\beta} m(\alpha \boldsymbol{g} + \beta B^{-1}\boldsymbol{g}), \qquad \|\alpha \boldsymbol{g} + \beta B^{-1}\boldsymbol{g}\|_2 \leq \Delta,$$

which is analytically tractable and has an exact solution for α and β.

8.5.2 When B has zero eigenvalues but no negative eigenvalues

When B has a zero eigenvalue we choose $\boldsymbol{p} = \boldsymbol{p}_{\mathrm{Cauchy}}$.

8.5.3 When B has negative eigenvalues

Then, we replace the spanning space in Equation (8.7) with:

$$\mathrm{Span}\left[\boldsymbol{g}, (B + \alpha\mathbb{I})^{-1}\boldsymbol{g}\right],$$

where α is a positive number in $(-\lambda_1, -2\lambda_1]$, and where λ_1 denotes the most negative eigenvalue of B. Thus, $B + \alpha\mathbb{I}$ is positive definite. Furthermore, we look at two sub-cases.

Case 1: If $\|(B + \alpha\mathbb{I})^{-1}\boldsymbol{g}\|_2 \leq \Delta$, then we expect that α should behave as $\alpha \gtrsim \|\boldsymbol{g}\|_2/\Delta$, in which case the corrected matrix is 'too far away' from the original matrix. We then discard the sub-space

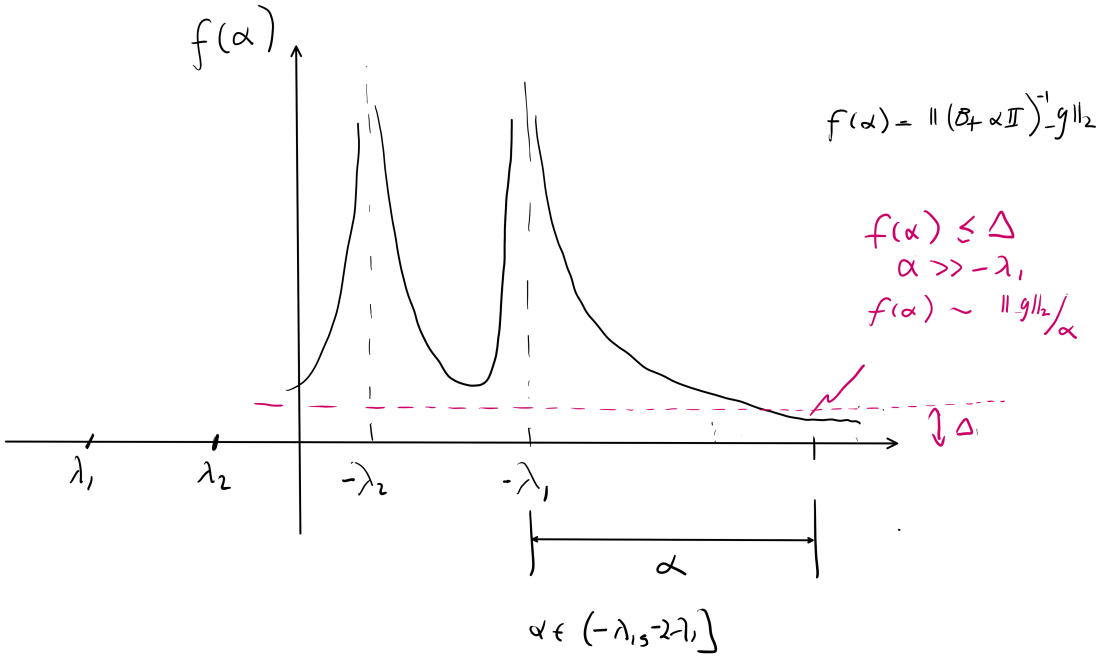


Figure 8.2: The idea behind Case 1 in the subspace minimization problem, in the case where B has negative eigenvalues. Here, for illustration, the eigenvalues λ_1 and λ_2 are taken to be negative, with λ_1 being the most negative eigenvalue.

minimization and define the step to be:

$$\boldsymbol{p} = -(B + \alpha\mathbb{I})^{-1}\boldsymbol{g} + \boldsymbol{v}, \tag{8.8}$$

where $\boldsymbol{v}$ is a vector that satisfies $\langle \boldsymbol{v}, (B + \alpha\mathbb{I})^{-1}\boldsymbol{g}\rangle \leq 0$.

This case can be understood better by looking at a particular example where λ_1 and λ_2 are both negative, and by considering the function

$$f(\alpha) = \|(B + \alpha\mathbb{I})^{-1}\boldsymbol{g}\|_2.$$

Then, the graph of $f(\alpha)$ is as shown in Figure 8.2. Thus, since α is restricted to positive values, if $f(\alpha) \leq \Delta$, then α is in the tail of the plot, where

$$f(\alpha) \sim \frac{\|\boldsymbol{g}\|_2}{\alpha}$$

In this case, α is large, and $B + \alpha\mathbb{I}$ is not a small perturbation of B, which justifies the alternative choice of search direction in Equation (8.8).

Case 2: If $\|(B + \alpha\mathbb{I})^{-1}\boldsymbol{g}\|_2 > \Delta$, then we expect that α should behave as $\alpha \lesssim \|\boldsymbol{g}\|_2/\Delta$ (see Figure 8.2 again), in which case the corrected matrix $B + \alpha\mathbb{I}$ is 'not too far away' from the original matrix B, and we proceed with the standard subspace minimization of $m(\boldsymbol{p})$ over $\boldsymbol{p} \in \mathrm{Span}\left[\boldsymbol{g}, (B + \alpha\mathbb{I})^{-1}\boldsymbol{g}\right]$, subject to $\|\boldsymbol{p}\|_2 \leq 1$.

Chapter 9

Analysis of the Quadratic Approximation

Overview

In trust-region methods, we are required to solve a constrained minimization problem ('the quadratic approximation') at each iteration, specifically,

$$\boldsymbol{p}_* = \arg\min_{\boldsymbol{p}} m(\boldsymbol{p}), \text{ subject to } \|\boldsymbol{p}\|_2 \leq \Delta, \tag{9.1}$$

where $m(\boldsymbol{p}) = \langle \boldsymbol{g}, \boldsymbol{p} \rangle + \frac{1}{2}\langle \boldsymbol{p}, B\boldsymbol{p} \rangle$ (we suppress the constant term in $m(\boldsymbol{p})$ for simplicity). We have previously stated necessary and sufficient conditions on the matrix B for a solution of Equation (9.1) to exist (Theorem 7.1), which we recall here as follows:

Theorem: Suppose that B is a symmetric matrix. Then the vector $\boldsymbol{p}_*$ is a global solution of the trust-region problem

$$\boldsymbol{p}_* = \arg\min_{\boldsymbol{p}} \left[\langle \boldsymbol{g}, \boldsymbol{p} \rangle + \tfrac{1}{2}\langle \boldsymbol{p}, B\boldsymbol{p} \rangle \right] \text{ subject to } \qquad \|\boldsymbol{p}\| \leq \Delta \tag{9.2}$$

if and only if $\boldsymbol{p}_*$ is feasible and there is a scalar λ such that the following conditions hold:

$$\begin{aligned}
\lambda &\geq 0, && \text{(9.3a)}\\
(B + \lambda \mathbb{I})\, \boldsymbol{p}_* &= -\boldsymbol{g}, && \text{(9.3b)}\\
\lambda\,(\Delta - \|\boldsymbol{p}_*\|_2) &= 0, && \text{(9.3c)}\\
(B + \lambda \mathbb{I}) &\quad \text{is positive semi-definite.} && \text{(9.3d)}
\end{aligned}$$

The aim of this section is to prove this Theorem. Then, inspired by some of the techniques used

to prove the Theorem, we will look at numerical methods to solve the constrained minimization problem for $m_k(\boldsymbol{p})$ in Equation (9.1) numerically.

9.1 Proof of the Theorem

Step 1: We begin by assuming that the conditions (9.3) hold. We introduce the auxiliary quadratic form $\widehat{m}(\boldsymbol{p})$:

$$\widehat{m}(\boldsymbol{p}) = \langle \boldsymbol{g}, \boldsymbol{p} \rangle + \tfrac{1}{2} \langle \boldsymbol{p}, (B + \lambda \mathbb{I})\, \boldsymbol{p} \rangle = m(\boldsymbol{p}) + \tfrac{1}{2} \lambda \langle \boldsymbol{p}, \boldsymbol{p} \rangle.$$

As $(B+\lambda\mathbb{I})$ is positive semi-definite and as $-\boldsymbol{g}$ can be written as $(B+\lambda\mathbb{I})\boldsymbol{p}_* = -\boldsymbol{g}$, by Theorem 2.10 $\widehat{m}(\boldsymbol{p})$ has a global minimizer, which we call $\boldsymbol{p}_*$:

$$\boldsymbol{p}_* = \arg\min \widehat{m}(\boldsymbol{p}),$$

hence

$$\widehat{m}(\boldsymbol{p}) \geq \widehat{m}(\boldsymbol{p}_*), \qquad \text{for all } \boldsymbol{p} \in \mathbb{R}^n.$$

In other words,

$$m(\boldsymbol{p}) + \tfrac{1}{2} \lambda \langle \boldsymbol{p}, \boldsymbol{p} \rangle \geq m(\boldsymbol{p}_*) + \tfrac{1}{2} \lambda \langle \boldsymbol{p}_*, \boldsymbol{p}_* \rangle, \qquad \text{for all } \boldsymbol{p} \in \mathbb{R}^n.$$

Hence,

$$m(\boldsymbol{p}) \geq m(\boldsymbol{p}_*) + \tfrac{1}{2} \lambda \left[\|\boldsymbol{p}_*\|_2^2 - \|\boldsymbol{p}\|_2^2 \right]. \tag{9.4}$$

By Equation (9.3)(c), we have $\lambda(\Delta - \|\boldsymbol{p}_*\|_2) = 0$, hence

$$\lambda(\Delta - \|\boldsymbol{p}_*\|_2)(\Delta + \|\boldsymbol{p}_*\|_2) = 0,$$

hence

$$\lambda(\Delta^2 - \|\boldsymbol{p}_*\|_2^2) = 0,$$

hence $\lambda\Delta^2 = \lambda\|\boldsymbol{p}_*\|_2^2$, hence Equation (9.4) becomes

$$m(\boldsymbol{p}) \geq m(\boldsymbol{p}_*) + \tfrac{1}{2} \lambda \left[\Delta^2 - \|\boldsymbol{p}\|_2^2 \right].$$

As we are constrained by $\|\boldsymbol{p}\|_2 \leq \Delta$, we have $\Delta^2 - \|\boldsymbol{p}\|_2^2 \geq 0$, hence

$$m(\boldsymbol{p}) \geq m(\boldsymbol{p}_*), \qquad \text{for all } \boldsymbol{p} \in \mathbb{R}^n,$$

hence $\boldsymbol{p}_*$ is a minimizer for the constrained problem (9.2).

Step 2: We next assume that Equation (9.2) has a solution, and we seek to show that conditions (9.3) hold. Denote the solution by $\boldsymbol{p}_*$. If $\|\boldsymbol{p}_*\|_2 < \Delta$, then we can treat the optimization problem (9.2) as effectively an unconstrained problem and hence, by Theorem (2.10), the conditions (9.3) hold with $\lambda = 0$.

Otherwise, we take $\|\boldsymbol{p}_*\|_2 = \Delta$ (we can adjust the parameter λ so that this is the case). Then, Equation (9.3) (**Part (c)**) is satisfied. To show that the other parts are true, we introduce the constrained problem

$$\mathcal{L}(\boldsymbol{p}, \lambda) = m(\boldsymbol{p}) + \tfrac{1}{2}\lambda \left[\|\boldsymbol{p}\|_2^2 - \Delta^2\right].$$

The minimum if this problem is $\boldsymbol{p}_*$. But by the theory of Lagrange multipliers, the minimum must satisfy $\nabla_{\boldsymbol{p}}\mathcal{L} = 0$, hence:

$$(B + \lambda\mathbb{I})\boldsymbol{p}_* = -\boldsymbol{g},$$

hence (**Part (b)**) is satisfied.

Thus, for all vectors $\boldsymbol{p}$ such that $\|\boldsymbol{p}\|_2 = \Delta$, we have:

$$m(\boldsymbol{p}) + \tfrac{1}{2}\lambda \left[\|\boldsymbol{p}\|_2^2 - \Delta^2\right] \geq m(\boldsymbol{p}_*) + \tfrac{1}{2}\lambda \left[\|\boldsymbol{p}_*\|_2^2 - \Delta^2\right].$$

or

$$m(\boldsymbol{p}) \geq m(\boldsymbol{p}_*) + \tfrac{1}{2}\lambda \left[\|\boldsymbol{p}_*\|_2^2 - \|\boldsymbol{p}\|_2^2\right].$$

Via repeated algebraic manipulations, this can be reduced to:

$$\langle \boldsymbol{p} - \boldsymbol{p}_*, (B + \lambda\mathbb{I})(\boldsymbol{p} - \boldsymbol{p}_*)\rangle \geq 0, \qquad \text{for all } \|\boldsymbol{p}\| = \Delta. \tag{9.5}$$

Consider now the set of vectors

$$X = \{\boldsymbol{\xi} | \boldsymbol{\xi} = \frac{\boldsymbol{p} - \boldsymbol{p}_*}{\|\boldsymbol{p} - \boldsymbol{p}_*\|_2}, \|\boldsymbol{p}\|_2 = \Delta\}.$$

By Equation (9.5), we have:

$$\langle \boldsymbol{\xi}, (B + \lambda\mathbb{I})\boldsymbol{\xi}\rangle \geq 0, \qquad \text{for all } \boldsymbol{\xi} \in X.$$

As the set X is dense in the unit sphere, we have

$$\langle \boldsymbol{\xi}, (B + \lambda\mathbb{I})\boldsymbol{\xi}\rangle \geq 0, \qquad \text{for all } \|\boldsymbol{\xi}\|_2 = 1.$$

and hence,

$$\langle \boldsymbol{\xi}, (B + \lambda\mathbb{I})\boldsymbol{\xi}\rangle \geq 0, \qquad \text{for all } \boldsymbol{\xi} \in \mathbb{R}^n,$$

and thus, $(B + \lambda\mathbb{I})$ is positive semi-definite, which confirms **Part (d)**.

It remains to show that $\lambda \geq 0$. We look at two sub-cases.

First Sub-Case: B is positive-semi-definite. As we have now established Parts (b)–(d), it $(B+\lambda\mathbb{I})$ is positive semi-definite, hence $\lambda \geq 0$.

First Sub-Case: B is not positive-semi-definite, so there is at least one non-zero vector $\boldsymbol{\xi}$ such that $\langle \boldsymbol{\xi}, B\boldsymbol{\xi} \rangle < 0$.

Assume for contradiction that $\lambda < 0$. As we have now established Parts (b)–(d), it follows that $(B + \lambda\mathbb{I})$ is positive semi-definite, hence

$$\begin{aligned} \langle \boldsymbol{\xi}, B\boldsymbol{\xi} \rangle &= -\lambda \langle \boldsymbol{\xi}, \boldsymbol{\xi} \rangle, \\ &= |\lambda| \langle \boldsymbol{\xi}, \boldsymbol{\xi} \rangle, \\ &< 0 \quad , \text{ for all } \boldsymbol{\xi} \neq 0. \end{aligned}$$

But this is a contradiction, hence $\lambda \geq 0$. Thus, Parts (a)–(d) are established. ∎

9.2 Numerical Techniques

We now sketch out a numerical technique for the solution of the Equation (9.2), which makes use of Theorem 7.1. If B^{-1} exists, and $-B^{-1}\boldsymbol{g}$ is in the trust region, then obviously, the solution to Equation (9.2) is just

$$\boldsymbol{p} = -B^{-1}\boldsymbol{g},$$

which is just the Newton step. Otherwise, we have to introduce a candididate solution

$$\boldsymbol{p}(\lambda) = -(B + \mathbb{I}\lambda)^{-1}\boldsymbol{g},$$

and we adjust λ so that this inverse matrix exists and also, such that

$$\|\boldsymbol{p}(\lambda)\|_2 = \Delta$$

(this is very much in the spirit of the adjustable parameter α in the approximate Dogleg method, in Chapter 8). We now outline how λ can be chosen.

At a minimum we assume that B is a symmetric matrix, meaning there is a complete set of mutually orthogonal eigenvectors $\boldsymbol{u}_i$,

$$B\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i, \qquad \langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle = \delta_{ij},$$

hence

$$\boldsymbol{p}(\lambda) = -\sum_{i=1}^{n} \frac{\langle \boldsymbol{g}, \boldsymbol{u}_i \rangle}{\lambda_i + \lambda} \boldsymbol{u}_i.$$

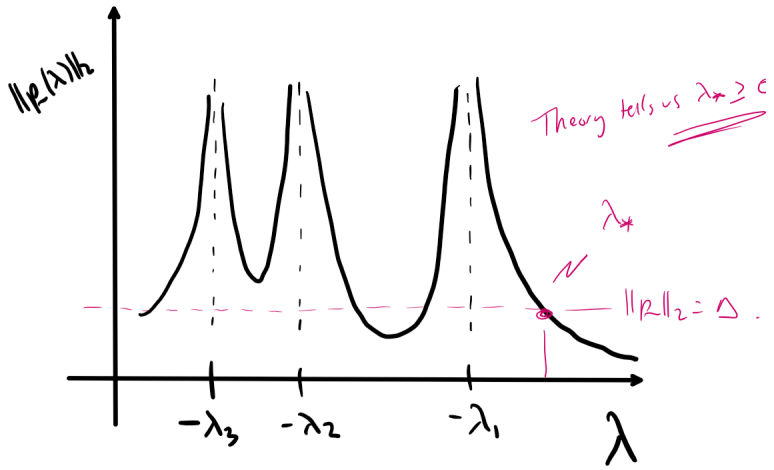


Figure 9.1: Finding the positive root λ_* of $\|\boldsymbol{p}(\lambda)\|_2 = \Delta$.

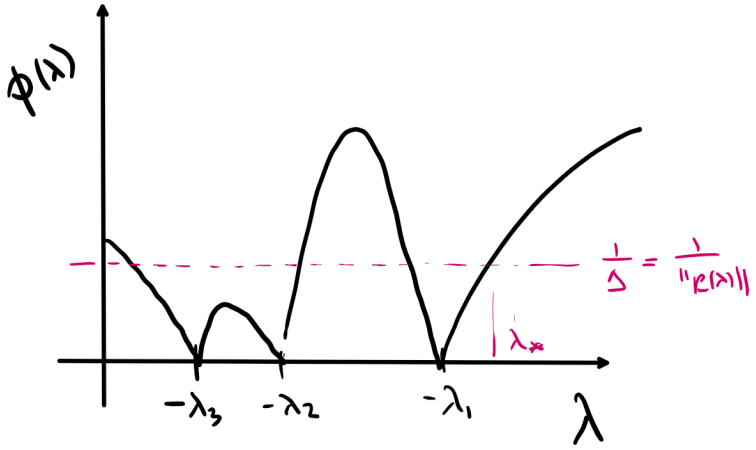


Figure 9.2: Finding the positive root λ_* of $\|\boldsymbol{p}(\lambda)\|_2 = \Delta$ without having to evaluate a function with singularities.

We assume an ordering $\lambda_1 \leq \lambda_2 \cdots \leq \lambda_n$. Thus,

$$\|\boldsymbol{p}(\lambda) = \sum_{i=1}^{n} \frac{|\langle \boldsymbol{g}, \boldsymbol{u}_i \rangle|^2}{(\lambda_i + \lambda)^2}. \tag{9.6}$$

This is a simple equation in λ, however, there are singularities when λ hits minus an eigenvalue, i.e. whenever $\lambda = -\lambda_i$. If we let λ denote the most-negative eigenvalue, then by a graphical argument (e.g. Figure 9.1), the root-finding condition $\|\boldsymbol{p}(\lambda)\|_2 = \Delta$ has a positive solution $\lambda_* > -\lambda_1$.

In practice, it is not nice to deal with a function with singularities, so when we are doing numerical root-finding, we look at the roots of

$$\phi(\lambda) = \frac{1}{\Delta} - \frac{1}{\|\boldsymbol{p}(\lambda)\|_2} \tag{9.7}$$

e.g. Figure 9.2. Thus, to find λ_* we use a Newton method, where the ℓ^{th} guess is updated to the

$(\ell+1)^{\text{th}}$ guess via:

$$\lambda^{(\ell+1)} = \lambda^{(\ell)} - \frac{\phi(\lambda^{(\ell)}}{\phi'(\lambda^{(\ell)})}.$$

In fact, the derivative combination $\phi(\lambda)/\phi'(\lambda)$ can be computed exactly by performing a Cholsesky decomposition on the matrix $B + \lambda\mathbb{I}$, which we assume to be symmetric and positive-definite: if $B + \lambda\mathbb{I} = R^T R$, and $(B + \lambda\mathbb{I})\boldsymbol{p} = -\boldsymbol{g}$, we introduce$\boldsymbol{q} = R^{-T}\boldsymbol{p}$, and then:

$$\frac{\phi(\lambda}{\phi'(\lambda)} = -\left(\frac{\|\boldsymbol{p}\|^2}{\|\boldsymbol{q}\|_2}\right)^2 \left(\frac{\|\boldsymbol{p}\| - \Delta}{\Delta}\right).$$

We then obtain the following Newton–Raphson algorithm for the computation of λ_*:

Algorithm 6 Computing λ_*

Choose an initial guess $\lambda^{(0)}$ for λ_* and a trust-region size $\Delta > 0$.

for $\ell = 0, 1, 2, \cdots$ **do**

Factor $B + \lambda^{\ell}\mathbb{I} = R^T R$;

Solve $R^T R\boldsymbol{p}_\ell = -\boldsymbol{g}$ and $R^T\boldsymbol{q}_\ell = \boldsymbol{p}_\ell$.

Set

$$\lambda^{(\ell+1)} = \lambda^{(\ell)} + \left(\frac{\|\boldsymbol{p}_\ell\|^2}{\|\boldsymbol{q}_\ell\|_2}\right)^2 \left(\frac{\|\boldsymbol{p}_\ell\| - \Delta}{\Delta}\right).$$

end for

Once a few more details are added to the algorithm to make sure that $\lambda^{(\ell)}$ goes not go below $-\lambda_1$, the algorithm does converge to a solution of Equation (9.7) (Nocedal and Wright say 'in most cases').

9.2.1 The Hard Case

There is an exceptional 'hard case' where the analysis breaks down. By inspection of Equation (9.6), this occurs when

$$\langle \boldsymbol{u}_1, \boldsymbol{g}\rangle = 0,$$

furthermore, this 'hard case' requires $\lambda_1 < 0$. Then, there is no $\lambda_* \in (-\lambda_1, \infty)$ such that $\|\boldsymbol{p}(\lambda_*)\|_2 = \Delta$. But can choose $\lambda = -\lambda_1$ and get a solution. The idea here is that

$$\boldsymbol{p}(\lambda) = -\sum_{i=2}^{n} \frac{\langle \boldsymbol{g}, \boldsymbol{u}_i\rangle}{\lambda_i + \lambda}\boldsymbol{u}_i.$$

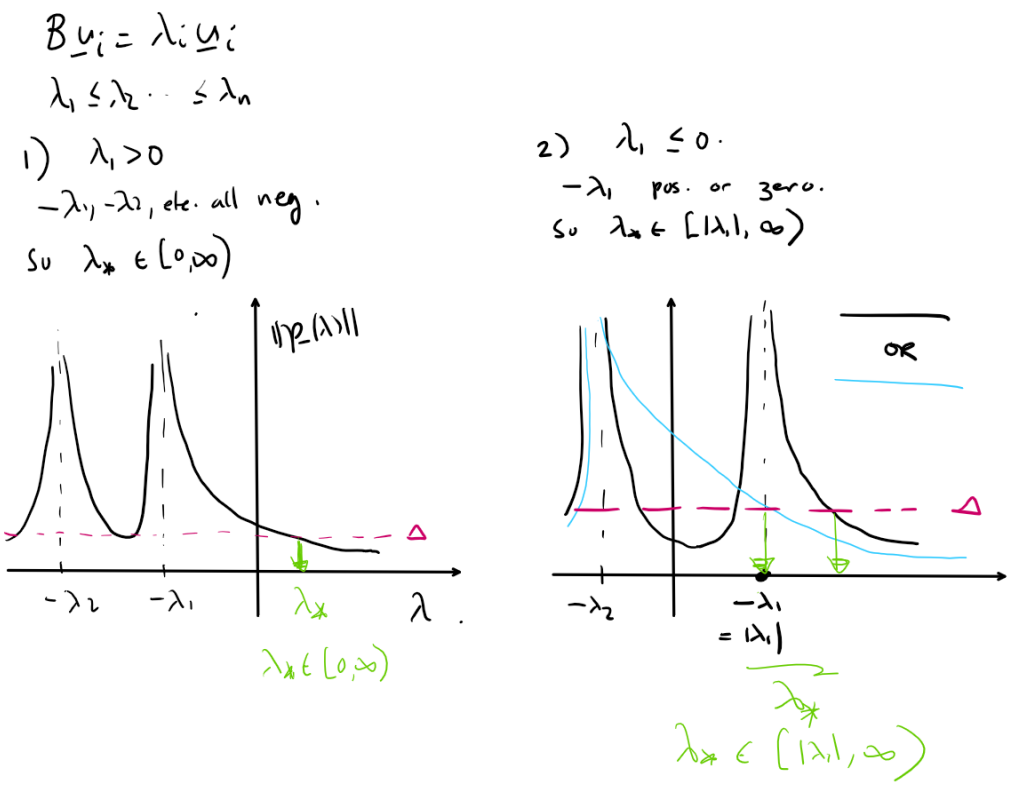


Figure 9.3: Plot of $\|\boldsymbol{p}(\lambda)\|_2$ versus λ showing the extraction of the root λ_* when $\|\boldsymbol{p}(\lambda_*)\|_2 = \Delta$. The hard case is shown in blue in Panel (b).

however, $(B - \lambda_1 \mathbb{I})\boldsymbol{u}_1 = 0$, hence

$$(B - \lambda_1 \mathbb{I}) \left[\tau \boldsymbol{u}_1 - \sum_{i=2}^{n} \frac{\langle \boldsymbol{g}, \boldsymbol{u}_i \rangle}{\lambda_i - \lambda_1} \boldsymbol{u}_i \right] = -\sum_{i=2}^{n} \langle \boldsymbol{g}, \boldsymbol{u}_i \rangle \boldsymbol{u}_i = -\boldsymbol{g},$$

hence

$$\boldsymbol{p}(\lambda) = \tau \boldsymbol{u}_1 - \sum_{i=2}^{n} \frac{\langle \boldsymbol{g}, \boldsymbol{u}_i \rangle}{\lambda_i - \lambda_1} \boldsymbol{u}_i,$$

where τ is an adjustable free parameter. We then choose τ such that $\|\boldsymbol{p}(\lambda)\|_2 = \Delta$.

The hard case specifically involves $\lambda_1 < 0$ as then we are concerned with the numerical problem of telling a numerical algorithm where to search for λ_*, specifically in the range $[-\lambda_*, \infty)$. If, on the other hand $\lambda_1 \geq 0$, then we can immediately apply Algorithm (9.2) to the search region $\lambda_* \in [0, \infty)$ and search for a solution that way. Figure 9.3 illustrates this idea.

9.3 Scaling again

Recall, we encountered badly-scaled problems in the context of the convergence analysis of the SD method. There, we identifyed poorly-scaled problems where the Hessian had eigenvalues such that

$$\lambda_{min} \ll \lambda_{max}. \tag{9.8}$$

In that case, we saw that the (linear) convergence of the SD method was severely degraded. There was no such problem with the Newton method. As it turns out, the trust-region methods exhibit poor scaling again in cases where the Hessian has the property (9.8). The solution here is to define a scaled descent direction,

$$\widetilde{\boldsymbol{p}} = D\boldsymbol{p},$$

where D is a diagonal **scaling matrix**. Thus, $\boldsymbol{p} = D^{-1}\widetilde{\boldsymbol{p}}$, which we substitute into the quadratic approximation: (e.g. Equation (7.5)). We recast this as a problem in terms of the scaled vector $\widetilde{\boldsymbol{p}}$:

$$\begin{aligned} \widetilde{\boldsymbol{p}}_* &= \arg\min_{\widetilde{\boldsymbol{p}}\in\mathbb{R}^n} \widetilde{m}_k(\widetilde{\boldsymbol{p}}), \\ &= \arg\min_{\widetilde{\boldsymbol{p}}\in\mathbb{R}^n} \left[f_k + \langle \boldsymbol{g}_k, D^{-1}\widetilde{\boldsymbol{p}}\rangle + \tfrac{1}{2}\widetilde{\boldsymbol{p}}, D^{-1}B_k D^{-1}\widetilde{\boldsymbol{p}}\rangle \right], \end{aligned}$$

which we solve subject to a scaled constraint

$$\|\widetilde{\boldsymbol{p}}\|_2 \leq \Delta.$$

Then, when this equation is solved for the scaled descent direction, the unscaled descent direction can be recovered via $\boldsymbol{p} = D^{-1}\widetilde{\boldsymbol{p}}$.

Chapter 10

Least-Squares Problems

Overview

In this section we look at Least-Squares Problems (which arise in Data Science) as a type of optimization. The numerical solution of such problems can be tackled with variations of the methods we have discussed previously, such as Line Search and Trust Region methods. In the case of so-called Linear Least Squares problems, the solution requires a matrix inversion; we show how to do this using Sinvular Value Decomposition.

10.1 Motivation

Suppose we make observations on a system where a measured quantity y depends on some input quantity t, through some unknown functional relationship. Suppose we make a large number of measurements m to yield measured values $\{y_1, \cdots, y_m\}$ at values $\{t_1, \cdots, t_n\}$ respectively. We then try to estimate the functional relationship between the y_i's and the t_i's. Suppose furthermore that we have some candidate for the relationship,

$$y(t) = \phi(t, \boldsymbol{x}),$$

where $\phi(t; \boldsymbol{x})$ is a function that depends on the input variable t but also, on parameters $\boldsymbol{x} = (x_1, \cdots, x_n)$, where $n < m$ is a relatively small number of parameters.

We would then have a statistical model for the relationship between the y_i's and the t_i's:

$$y_i(t_i) = \phi(t_i, \boldsymbol{x}) + \epsilon_i, \qquad i = 1, 2, \cdots, m,$$

where ϵ_i is some error, usually assumed to be drawn from m independent identical distributions. We

then introduce the cost function which is the deviation of the observations from the model:

$$f(\boldsymbol{x}) = \tfrac{1}{2}\sum_{i=1}^{m}\left[\phi(t_i;\boldsymbol{x}) - y_i\right]^2. \tag{10.1}$$

We know that if the error terms ϵ_i are drawn from a Guassian distribution with mean zero and variance σ, and distribution

$$g_\sigma(\epsilon) = \frac{1}{\sqrt{2\pi\sigma^2}}\mathrm{e}^{-\epsilon^2/2\sigma^2},$$

then minimizing the cost function $f(\boldsymbol{x})$ maximizes the likelihood function

$$p(y_1,\cdots,y_m;\boldsymbol{x},\sigma) = \prod_{j=1}^{m} g_\sigma(\epsilon_j) = \prod_{j=1}^{m} g_\sigma(\phi(\boldsymbol{x};t_j) = y_j),$$

and the corresponding minimum $\boldsymbol{x}_*$ is the **maximum likelihood estimate** for the parameters $\boldsymbol{x}$.

10.1.1 How this differs from previous chapters

The point of departure of this chapter is that the cost function in Equation (10.1) has a special structure, this enables us to apply special cases of the previously-studied optimization techniques and hence to develop robust and efficient methods to compute the minimum of the cost function. Specifically, we identify the **residual**

$$r_j(\boldsymbol{x}) = \phi(t_i;\boldsymbol{x}) - y_i, \qquad i = 1,2,\cdots,m$$

hence, the cost function (10.1) has the structure

$$f(\boldsymbol{x}) = \tfrac{1}{2}\sum_{i=1}^{m}[r_i(\boldsymbol{x})]^2. \tag{10.2}$$

10.2 Linear Least-Squares Problems

In the case where ϕ is a linear function of the parameters $\boldsymbol{x}$, then

$$\phi(t_i;\boldsymbol{x}) = \sum_{j=1}^{n} J_{ij}x_j, \qquad i = 1,2,\cdots,m,$$

hence J is an $m \times n$ matrix (not square!). Thus, the cost function (10.1) becomes:

$$f(\boldsymbol{x}) = \tfrac{1}{2}\|J\boldsymbol{x} - \boldsymbol{y}\|_2^2, \tag{10.3}$$

where $\boldsymbol{y} \in \mathbb{R}^m$ is an m-dimensional column vector. Thus,

$$\begin{aligned} f(\boldsymbol{x}) &= \tfrac{1}{2}\langle J\boldsymbol{x} - \boldsymbol{y}, J\boldsymbol{x} - \boldsymbol{y}\rangle, \\ &= \tfrac{1}{2}\langle J\boldsymbol{x}, J\boldsymbol{x}\rangle - \langle J\boldsymbol{x}, \boldsymbol{y}\rangle + \tfrac{1}{2}\langle \boldsymbol{y}, \boldsymbol{y}\rangle, \\ &= \tfrac{1}{2}\langle \boldsymbol{x}, J^T J\boldsymbol{x}\rangle - \langle J^T\boldsymbol{y}, \boldsymbol{x}\rangle + \tfrac{1}{2}\langle \boldsymbol{y}, \boldsymbol{y}\rangle. \end{aligned}$$

The first-order optimality condition $\nabla f = 0$ at $\boldsymbol{x} = \boldsymbol{x}_*$ then gives:

$$J^T J\boldsymbol{x}_* = J^T\boldsymbol{y} \tag{10.4}$$

Equation (10.4) is called the **normal equation**.

10.3 Solution of the Normal Equation

If J has full column rank, then Equation (10.4) can be solved numerically via matrix inversion. However, even the choice of matrix inversion here is not clear, there at least three numerical methods which can be considered for the purpose of inverting the normal equation:

- Cholesky factorization of $J^T J$.
- QR factorization $J^T J$.
- Singular Value Decomposition (SVD).

We look at the third method here briefly, it has an advantage in that it can be used even in the case where J does not have full column rank.

10.3.1 SVD

We apply the SVD to the matrix J:

$$J = \underbrace{U}_{m\times m} \underbrace{\begin{bmatrix} \sigma_1 & 0 & \cdots & 0, \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & \sigma_n \\ & 0 & 0 & \uparrow \\ & \vdots & \vdots & m-n \\ & & & \downarrow \end{bmatrix}}_{=\Sigma} \underbrace{V^T}_{n\times n}.$$

where $\sigma_1, \cdots$ are the singular values of the matrix J, and U and V are orthogonal matrices such that $U^TU = \mathbb{I}_{m\times m}$ and $V^TV = \mathbb{I}_{n\times n}$. Furthermore,

$$\begin{aligned} J^TJ &= V\Sigma^TU^TU\Sigma V^T, \\ &= V(\Sigma^T\Sigma)V^T. \end{aligned}$$

We call the matrix $\mathrm{diag}(\sigma_1, \cdots, \sigma_n) = S$, hence

$$\Sigma = \begin{bmatrix} S \\ 0 \end{bmatrix}$$

where the zeros take up $m - n$ rows. Hence, $\Sigma^T\Sigma = S^2$ and

$$J^TJ = VS^2V^T. \tag{10.5}$$

However, J^TJ is a symmetric matrix in $\mathbb{R}^{n\times n}$, as such, it can also be written as:

$$J^TJ = V \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & \lambda_n \end{pmatrix} V^T, \tag{10.6}$$

where we identify:

$$J^TJ\boldsymbol{u}_i = \lambda_i\boldsymbol{u}_i, \qquad i = 1, 2, \cdots, n, \qquad \lambda_i \in \mathbb{R},$$

and hence,

$$V = \begin{pmatrix} \uparrow & \cdots & \uparrow \\ \boldsymbol{u}_1 & \cdots & \boldsymbol{u}_n \\ \downarrow & & \downarrow \end{pmatrix}$$

Furthermore, the λ_i's are non-negative in view of the structure of the matrix J^TJ. Comparing Equations (10.5) and (10.6), we have

$$\sigma_i = \sqrt{\lambda_i}.$$

We now look at the cost function (10.3) in more detail. We have:

$$\begin{aligned} 2f(\boldsymbol{x}) &= \|J\boldsymbol{x}-\boldsymbol{y}\|_2^2, \\ &= \|U\begin{bmatrix} S \\ 0 \end{bmatrix}V^T\boldsymbol{x} - \underbrace{UU^T}_{=\mathbb{I}_{n\times n}}\boldsymbol{y}\|_2^2, \\ &= \|U\left\{\begin{bmatrix} S \\ 0 \end{bmatrix}V^T\boldsymbol{x} - U^T\boldsymbol{y}\right\}\|_2^2, \\ &\overset{\text{Orthogonal}}{=} \|\begin{bmatrix} S \\ 0 \end{bmatrix}V^T\boldsymbol{x} - U^T\boldsymbol{y}\|_2^2. \end{aligned}$$

Let

$$\mathbb{R}^{m\times m} \ni U = \begin{bmatrix} \underbrace{U_1}_{n \text{ cols}} & \underbrace{U_2}_{(m-n) \text{ cols}} \end{bmatrix} \updownarrow m \text{ rows}$$

Hence,

$$U^T = \begin{bmatrix} U_1^T \\ U_2^T \end{bmatrix}.$$

Thus, the cost function becomes:

$$\begin{aligned} 2f(\boldsymbol{x}) &= \|\begin{bmatrix} S \\ 0 \end{bmatrix}V^T\boldsymbol{x} - \begin{bmatrix} U_1^T \\ U_2^T \end{bmatrix}\boldsymbol{y}\|_2^2, \\ &= \|\begin{bmatrix} SV^T\boldsymbol{x} - U_1^T\boldsymbol{y} \\ -U_2^T\boldsymbol{y} \end{bmatrix}\|_2^2 \end{aligned}$$

Just as $\langle(x_1,x_2),(x_1,x_2)\rangle = x_1^2+x_2^2$ for a two-dimensional vector, the same principle applies in higher-dimensional space where the vector can be partitioned into two blocks. Hence, $2f(\boldsymbol{x})$ becomes:

$$2f(vecx) = \|SV^T\boldsymbol{x} - U_1^T\boldsymbol{y}\|_2^2 + \|U_2^T\boldsymbol{y}\|_2^2. \tag{10.7}$$

Clearly, to minimize $2f(\boldsymbol{x})$ and hence, to solve the normal equation (10.4), we require:

$$SV^T\boldsymbol{x}_* = U_1^T\boldsymbol{y}.$$

We look at two cases.

10.3.2 S has no zero entries on the diagonal

In this case,

$$\boldsymbol{x}_* = VS^{-1}U_1^T\boldsymbol{y},$$

hence:

$$\boldsymbol{x}_* = \overbrace{\begin{pmatrix} | & \cdots & | \\ \boldsymbol{v}_1 & & \boldsymbol{v}_n \\ | & & | \end{pmatrix}}^{n\times n} \overbrace{\begin{pmatrix} 1/\sigma_1 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 1/\sigma_n \end{pmatrix}}^{n\times n} \overbrace{\begin{pmatrix} -- & \boldsymbol{u}_1 & -- \\ -- & \vdots & -- \\ -- & \boldsymbol{u}_n & -- \end{pmatrix}}^{n\times m} \overbrace{\begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}}^{m\times 1}$$

Hence,

$$\begin{aligned} \boldsymbol{x}_* &= \begin{pmatrix} | & \cdots & | \\ \boldsymbol{v}_1 & & \boldsymbol{v}_n \\ | & & | \end{pmatrix} \begin{pmatrix} 1/\sigma_1 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 1/\sigma_n \end{pmatrix} \begin{pmatrix} \langle \boldsymbol{u}_1, \boldsymbol{y} \rangle \\ \vdots \\ \langle \boldsymbol{u}_n, \boldsymbol{y} \rangle \end{pmatrix}, \\ &= \begin{pmatrix} | & \cdots & | \\ \boldsymbol{v}_1 & & \boldsymbol{v}_n \\ | & & | \end{pmatrix} \begin{pmatrix} \langle \boldsymbol{u}_1, \boldsymbol{y} \rangle / \sigma_1 \\ \vdots \\ \langle \boldsymbol{u}_n, \boldsymbol{y} \rangle / \sigma_n \end{pmatrix} \end{aligned}$$

We perform the remaining matrix multiplication index-wise, and we use $\boldsymbol{v}_i^{(j)}$ to denote the i^{th} entry in the j^{th} eigenvector $\boldsymbol{v}_j$:

$$(\boldsymbol{x}_*)_i = \sum_{j=1}^{n} v_i^{(j)} \frac{\langle \boldsymbol{u}_j, \boldsymbol{y} \rangle}{\sigma_j},$$

hence

$$\boldsymbol{x}_* = \sum_{j=1}^{n} \boldsymbol{v}_j \frac{\langle \boldsymbol{u}_j, \boldsymbol{y} \rangle}{\sigma_j},$$

which is the required solution of the normal equation (10.4).

10.3.3 S has zero entries on the diagonal

In this case, J is rank-deficient, and a (non-unique) solution of the minimization problem is given by:

$$\boldsymbol{x}_* = \sum_{\sigma_j \neq 0} \boldsymbol{v}_j \frac{\langle \boldsymbol{u}_j, \boldsymbol{y} \rangle}{\sigma_j} + \sum_{\sigma_j = 0} \tau_j \boldsymbol{v}_j.$$

- Experience (Nocdeal and Wright, Section 10.2) suggests that the non-unique solution with the smallest norm is best, hence $\tau_j = 0$.
- In cases where the σ_j's are all nonzero, but one or two of the σ_j's are close to zero (hence, a badly-scaled problem), then the solution $\boldsymbol{x}_*$ is very sensitive to slight changes in $\langle \boldsymbol{u}_j, \boldsymbol{y} \rangle$.
- In such cases, an approximate solution of the normal equation can be better, in which case these components are set to zero.

Chapter 11

Nonlinear Least Squares

Overview

We look at Nonlinear Least Squares problems, where the numerical solution requires the use of trust-region methods. As the cost function is bespoke to nonlienar least-squares problems, the tailor-made implementation of the trust-region methods is also bespoke, this goes by the name of the **Levenberg–Marquart** method.

11.1 The Algebra

We start with the least-squares cost function

$$2f(\boldsymbol{x}) = \sum_{\alpha=1}^{m} [r_\alpha(\boldsymbol{x})]^2, \qquad \boldsymbol{x} \in \mathbb{R}^n.$$

Hence,

$$\nabla_x f = \sum_{\alpha=1}^{m} r_\alpha(\boldsymbol{x}) \nabla_x r_\alpha(\boldsymbol{x}),$$

or

$$\frac{\partial f}{\partial x_i} = \sum_{\alpha=1}^{m} r_\alpha(\boldsymbol{x}) \frac{\partial r_\alpha}{\partial x_i}.$$

Consider

$$\begin{pmatrix} \uparrow \\ \nabla f \\ \downarrow \end{pmatrix} = \overbrace{\begin{pmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_2}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_1} \\ \frac{\partial r_1}{\partial x_2} & \frac{\partial r_2}{\partial x_2} & \cdots & \frac{\partial r_m}{\partial x_2} \\ \vdots & & & \vdots \\ \frac{\partial r_1}{\partial x_n} & \frac{\partial r_2}{\partial x_n} & \cdots & \frac{\partial r_m}{\partial x_n} \end{pmatrix}}^{\leftarrow m \rightarrow} \begin{pmatrix} r_1 \\ \cdots \\ r_m \end{pmatrix} \begin{matrix} \uparrow \\ n \\ \downarrow \end{matrix}$$

We identify $J \in \mathbb{R}^{m\times n}$. Hence,

$$\mathbb{R}^{n\times m} = J^T = \begin{pmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_2}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_1} \\ \frac{\partial r_1}{\partial x_2} & \frac{\partial r_2}{\partial x_2} & \cdots & \frac{\partial r_m}{\partial x_2} \\ \vdots & & & \vdots \\ \frac{\partial r_1}{\partial x_n} & \frac{\partial r_2}{\partial x_n} & \cdots & \frac{\partial r_m}{\partial x_n} \end{pmatrix},$$

and

$$J = \begin{pmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} & \cdots & \frac{\partial r_1}{\partial x_n} \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} & \cdots & \frac{\partial r_2}{\partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial r_m}{\partial x_1} & \frac{\partial r_m}{\partial x_2} & \cdots & \frac{\partial r_m}{\partial x_n} \end{pmatrix}$$

and finally,

$$J_{ij} = \frac{\partial r_i}{\partial x_j}, \qquad i \in \{1, 2, \cdots, m\}, \qquad j \in \{1, 2, \cdots, n\}.$$

Hence,

$$\begin{aligned} (\nabla f)_i &= \sum_{\alpha=1}^{m} r_\alpha \frac{\partial r_\alpha}{\partial x_i}, \\ &= \sum_{\alpha=1}^{m} J_{\alpha i} r_\alpha, \\ &= \sum_{\alpha=1}^{m} (J^T)_{i\alpha} r_\alpha, \\ &= (J^T \boldsymbol{r}), \end{aligned}$$

where $\boldsymbol{r} = (r_1, \cdots, r_m)^T$. Hence,

$$\nabla f = J^T \boldsymbol{r}.$$

11.2 The Hesisan Matrix

We have:

$$\begin{aligned} \frac{\partial^2 f}{\partial x_i \partial x_j} &= \frac{\partial}{\partial x_i} \sum_{\alpha=1}^{m} r_\alpha(\boldsymbol{x}) \frac{\partial r_\alpha}{\partial x_j}, \\ &= \sum_{\alpha=1}^{m} r_\alpha(\boldsymbol{x}) \frac{\partial^2 r_\alpha}{\partial x_i \partial x_j} + \sum_{\alpha=1}^{m} \frac{\partial r_\alpha}{\partial x_i} \frac{\partial r_\alpha}{\partial x_j}. \end{aligned}$$

Consider

$$\begin{aligned}(J^TJ)_{ij} &= \sum_{\alpha=1}^{m}(J^T)_{i\alpha}J_{\alpha j},\\ &= \sum_{\alpha=1}^{m}J_{\alpha i}J_{\alpha j},\\ &= \sum_{\alpha=1}^{m}\frac{\partial r_\alpha}{\partial x_i}\frac{\partial r_\alpha}{\partial x_j}.\end{aligned}$$

Hence,

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = (J^TJ)_{ij} + \sum_{\alpha=1}^{m} r_\alpha(\boldsymbol{x})\frac{\partial^2 r_\alpha}{\partial x_i \partial x_j}.$$

Thus, if the residuals are small, then J^TJ is a godo approximation to the Hessian. The approximation is exact in the case of a linear model.

11.3 Gauss–Newton Method

The Gauss–Newton is a line-search method where the descent direction where the update step is given by:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_{\mathrm{GN}}. \tag{11.1}$$

The search direction $\boldsymbol{p}_{\mathrm{GN}}$ is computed in a way that is inspired by the Newton method, but instead of $B\boldsymbol{p} = -\boldsymbol{g}$, where B is the Hessian matrix, we use the approximate Hessian J^TJ to compute:

$$J_k^TJ_k\boldsymbol{p}_{\mathrm{GN}} = -\boldsymbol{g}, \qquad \boldsymbol{g}_k = J_k\boldsymbol{r}_k. \tag{11.2}$$

Hence,

$$J_k^TJ_k\boldsymbol{p}_{\mathrm{GN}} = -\nabla f_k. \tag{11.3}$$

We now check that $\boldsymbol{p}_{\mathrm{GN}}$ is indeed a descent direction. We suppress the iteration index and we compute:

$$\begin{aligned}\langle \nabla f, \boldsymbol{p}_{\mathrm{GN}}\rangle &= \langle -J^TJ\boldsymbol{p}_{\mathrm{GN}}, \boldsymbol{p}_{\mathrm{GN}}\rangle,\\ &= -\langle J\boldsymbol{p}_{\mathrm{GN}}, J\boldsymbol{p}_{\mathrm{GN}}\rangle,\\ &= -\|J\boldsymbol{p}_{\mathrm{GN}}\|_2^2.\end{aligned}$$

If J has full rank, and if $\nabla f \neq 0$, then

$$\langle \nabla f, \boldsymbol{p}_{\mathrm{GN}}\rangle < 0. \tag{11.4}$$

11.3.1 Convergence Analysis

To make progress here, we introduce some notation. We let $\boldsymbol{x}_0$ be the starting-point of the interative method (11.1). We define the set

$$\mathcal{L} = \{\boldsymbol{x} | f(\boldsymbol{x}) \leq f(\boldsymbol{x}_0)\},$$

and we let $\mathcal{N}$ denote an open neighbourhood in $\mathcal{L}$. We say that the matrix J is bounded away from zero in $\mathcal{N}$ if there exists a $\gamma > 0$ such that, for all $\boldsymbol{x} \in \mathcal{N}$, the following relation holds:

$$\|J(\boldsymbol{x})\boldsymbol{z}\|_2 \geq \gamma \|\boldsymbol{z}\|_2, \qquad \text{for all } \boldsymbol{z} \neq 0 \text{ in } \mathbb{R}^n. \tag{11.5}$$

As Equation (11.5) is true for all $\boldsymbol{z} \in \mathbb{R}^n$, it follows that

$$\lambda_{min} \geq \gamma,$$

hence where λ_{min} is the minimum eigenvalue of $J^T J$. Thus, $J^T J$ is positive definite and it follows that J itself has full rank. Thus, Equation (11.5) is also called the full-rank condition.

We now prove the following theorem.

Theorem 11.1 *Suppose that each residual function $r_\alpha(\boldsymbol{x})$ in the cost function*

$$f(\boldsymbol{x}) = \tfrac{1}{2} \sum_{\alpha=1}^{m} [r_\alpha(\boldsymbol{x})]^2$$

is Lipschitz continuously differentiable in a neighbourhood $\mathcal{N}$ of the bounded set $\mathcal{L}$. Suppose also that the Jacobians $J(\boldsymbol{x})$ satisfy the uniform full-rank condition (11.5) *on $\mathcal{N}$. Then, if the iterative method is given by*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_{\mathrm{GN}},$$

where each stepsize α_k satisfies the SWCS, we have:

$$\lim_{k \to \infty} J_k^T \boldsymbol{r}_k = 0.$$

Proof: By construction of the set $\mathcal{L}$, $f(\boldsymbol{x})$ is bounded for all $\boldsymbol{x} \in \mathcal{L}$. Thus, for the subset $\mathcal{N} \subset \mathcal{L}$, there exists a constant B_1 such that

$$|r_\alpha(\boldsymbol{x})| \leq B_1 \qquad \text{for all } \boldsymbol{x} \in \mathcal{L}.$$

By the Lipschitz property on r_α, it is easy to show that $\|\nabla_x r_\alpha(\boldsymbol{x})\|_2 \leq B_2$ for all $\boldsymbol{x} \in \mathcal{N}$. Hence,

by taking $\beta = \max(B_1, B_2)$, we have:

$$|r_\alpha(\boldsymbol{x})| \leq \beta, \qquad \|\nabla_x r_\alpha\|_2 \leq \beta, \qquad \text{for all } \boldsymbol{x} \in \mathcal{N}.$$

Furthermore, by the Lipschitz property on r_α and its derivatives, there exists a constant $L > 0$ such that:

$$|r_\alpha(\boldsymbol{x}) - r_\alpha(\widetilde{\boldsymbol{x}})| \leq L\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|_2, \qquad \|\nabla_x r_\alpha(\boldsymbol{x}) - \nabla_x r_\alpha(\widetilde{\boldsymbol{x}})\|_2 \leq L\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|_2, \qquad \text{for all } \boldsymbol{x} \text{ and } \widetilde{\boldsymbol{x}} \in \mathcal{N}.$$

As $J_{ij} = \partial r_i / \partial x_j$, it follows that $J_{ij}(\boldsymbol{x})$ is bounded on $\mathcal{N}$ and hence, $\|J(\boldsymbol{x})\|_2$ is bounded as well. Thus, there exists a constant $\widetilde{\beta}$ such that:

$$\|J(\boldsymbol{x})\|_2 \leq \widetilde{\beta} \text{ for all } \boldsymbol{x} \in \mathcal{N}.$$

Furthermore, as

$$(\nabla f)_i = \sum_{\alpha=1}^{m} r_\alpha(\boldsymbol{x}) \frac{\partial r_\alpha}{\partial x_i} \tag{11.6}$$

is the product of Lipschitz functions, ∇f itself is Lipschitz (on $\mathcal{L}$). We review:

- f is bounded below... by zero;
- f is continously differentiable on $\mathcal{N}$... because its derivatives are Lipschitz.
- ∇f is Lipschitz on $\mathcal{N}$... by Equation (11.6).

Thus, Theorem 6.1 applies: we have

$$\cos\theta_k = \frac{-\langle \nabla f_k, \boldsymbol{p}_{\mathrm{GN}} \rangle}{\|\nabla f_k\|_2 \|\boldsymbol{p}_{\mathrm{GN}}\|_2},$$

and

$$\sum_{k \geq 0} \cos^2\theta_k \|\nabla f_k\|_2^2 < \infty. \tag{11.7}$$

Furthermore,

$$\begin{aligned} \cos\theta_k &= \frac{\|J\boldsymbol{p}_{\mathrm{GN}}\|_2^2}{\|\boldsymbol{p}_{\mathrm{GN}}\|_2 \|J^T J \boldsymbol{p}_{\mathrm{GN}}\|_2}, \\ &\overset{\text{Eq. (11.5)}}{\geq} \frac{\gamma^2 \|\boldsymbol{p}_{\mathrm{GN}}\|_2^2}{\widetilde{\beta}\|\boldsymbol{p}_{\mathrm{GN}}\|_2^2}, \\ &= \frac{\gamma^2}{\widetilde{\beta}^2}, \\ &> 0. \end{aligned}$$

Hence, referring back to Equation (11.7), $\|\nabla f_k\|_2 \to 0$ as $k \to \infty$. ∎

11.4 Levenberg–Marquardt Method

The Levenberg–Marquardt method is an iterative method to solve the nonlinear least squares problem

$$\boldsymbol{x}_* = \arg\min f(\boldsymbol{x}), \qquad f(\boldsymbol{x}) = \tfrac{1}{2}\sum_{\alpha=1}^{m}[r_\alpha(\boldsymbol{x})]^2. \tag{11.8}$$

At a typical iteration $\boldsymbol{x}_k$, the nonlinear problem (11.8) is approximated using the quadratic approximation:

$$\begin{aligned}
f(\boldsymbol{x}) &= f(\boldsymbol{x}_k + \boldsymbol{p}),\\
&\approx f(\boldsymbol{x}_k) + \langle \nabla f, \boldsymbol{p}\rangle + \tfrac{1}{2}\sum_{i,j} p_i p_j \frac{\partial^2 f}{\partial x_i \partial x_j},\\
&\approx f(\boldsymbol{x}_k) + \langle \nabla f, \boldsymbol{p}\rangle + \tfrac{1}{2}\sum_{i,j} p_i p_j [J^T J](\boldsymbol{x}_k),\\
&= f(\boldsymbol{x}) + \langle J^T\boldsymbol{r}, \boldsymbol{p}\rangle + \tfrac{1}{2}\langle \boldsymbol{p}, J^T J\boldsymbol{p}\rangle,\\
&= \tfrac{1}{2}\langle \boldsymbol{r}_k, \boldsymbol{r}_k\rangle + \langle J^T\boldsymbol{r}, \boldsymbol{p}\rangle + \tfrac{1}{2}\langle \boldsymbol{p}, J^T J\boldsymbol{p}\rangle,\\
&= \tfrac{1}{2}\langle J_k\boldsymbol{p} + \boldsymbol{r}_k, J_k\boldsymbol{p} + \boldsymbol{r}_k\rangle,\\
&= \tfrac{1}{2}\|J_k\boldsymbol{p} + \boldsymbol{r}_k\|_2^2,\\
&= m_k(\boldsymbol{p}).
\end{aligned}$$

The descent direction is then extracted via the **trust-region method**:

$$\boldsymbol{p}_k = \arg\min_{\|\boldsymbol{p}\|_2 \le \Delta} m_k(\boldsymbol{p}). \tag{11.9}$$

As in Chapter 9, the solution of the **constrained** minimization problem 11.9 splits into two possibilities. First of all, the Gauss–Newton descent direction is computed, as the solution of $J^T J\boldsymbol{p}_{\mathrm{GN}} = -J^T\boldsymbol{r}$. Then,

1. If $\|\boldsymbol{p}_{\mathrm{GN}}\|_2 \le \Delta$, then $\boldsymbol{p}_k = \boldsymbol{p}_{\mathrm{GN}}$.
2. Otherwise, if $\|\boldsymbol{p}_{\mathrm{GN}}\|_2 > \Delta$, we solve

$$(J^T J + \lambda\mathbb{I})\boldsymbol{p} = -J^T\boldsymbol{r}, \tag{11.10}$$

and adjust the non-negative parameter λ such that $\|\boldsymbol{p}\|_2 = \Delta$.

In Case 2, there are a couple of comments:

- Solving Equation (11.10) is somewhat easier than the previous, analogous problem in Chapter 9, as $J^T J$ is already positive semi-definite.

- Equation (11.10) is solved numerically using Algorithm 9.2; now, however, 'fast methods' can be used for the Cholesky factorization $J^T J + \lambda \mathbb{I} = R^T R$, where R is an upper-triangular matrix.

Chapter 12

Introduction to Constrained Optimization

Overview

We outline some of the basic ideas in constrained optimization.

12.1 Introduction

In this part of the course, we seek solutions of the constrained optimization problem: Using this notation, the generic optimization problem to be studied in this module is:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \text{ subject to } \begin{cases} c_i(\boldsymbol{x}) = 0, & i \in \mathcal{E}, \\ c_i(\boldsymbol{x}) \geq 0, & i \in \mathcal{I}. \end{cases} \tag{12.1}$$

We recall the definition of the **feasible set** from Chapter 2:

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n | c_i(\boldsymbol{x}) = 0,\, i \in \mathcal{E},\ c_i(\boldsymbol{x}) \geq 0\, i \in \mathcal{I}\}. \tag{12.2}$$

The solution to the OP (12.1) is denoted by $\boldsymbol{x}_*$.

Constrained optimization is a detailed, technical subject. As such, in this part of the course, we only have time to derive a set of **necessary conditions**, such that $\boldsymbol{x}_*$ is a solution to the OP; these conditions will be called the **Kurush–Kuhn–Tucker** conditions.

We recall first of all the constrained case: if $\boldsymbol{x}_*$ is a local minimizer of the unconstrained optimization problem $\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x})$, then $\nabla f(\boldsymbol{x}_*) = 0$ and $B_{ij} = (\partial^2 f/\partial x_i \partial x_j)_{\boldsymbol{x}_*}$ is a positive semi-definite matrix. Furthermore, if

$$\nabla f(\boldsymbol{x}_*) = 0, \qquad B \text{ positive definite} \tag{12.3}$$

then $\boldsymbol{x}_*$ is a strong local minimizer of the unconstrained OP; the conditions (12.3) are then sufficient conditions for the existence of the strong local minimizer. Summarizing, and for the present purposes, the first-order necessary condition for $\boldsymbol{x}_*$ to be a local minimizer is simply:

$$\nabla f = 0, \qquad \text{at } \boldsymbol{x} = \boldsymbol{x}_*. \tag{12.4}$$

The aim of this part of the course is to obtain analogous conditions in the constrained case, for the OP (12.1).

12.1.1 Review of the different types of minimum

We first of all review the different types of minimum for an optimization problem, starting with a motivating example:

$$\min f(\boldsymbol{x}) = (y + 100)^2 + \tfrac{1}{100}x^2 \tag{12.5a}$$

subject to

$$y - \cos(x) \geq 0. \tag{12.5b}$$

The feasible set is shown in Figure 12.1. By inspection of $f(\boldsymbol{x})$ in Equation (12.5), the global minimum is at $x = 0$ and $y = -100$, but this is not in the feasible set. However, by 'making y as small as possible within a neighbourhood', we can find local solutions of the constrained optimization problem. Referring to the figure, we see that the local solutions occur at:

$$\boldsymbol{x}_k = (k\pi, -1), \qquad k = \pm 1, \pm 3, \cdots$$

This is for sure a **pathological example** where the addition of a constraint introduces a multiplicity of local minimizers. A more 'usual' occurrence is for the constraints to 'weed out' certain local minimizers of the unconstrained problem, this often makes the search for a global minimizer of the constrained problem that bit easier.

Motivated by this example, we classify the possible feasible solutions of the constrained OP as follows:

1. $\boldsymbol{x}_*$ is a local solution of the OP if:
   - $\boldsymbol{x}_* \in \Omega$,
   - There exists a neighbourhood $\boldsymbol{x}_* \ni \mathcal{N}$ such that:

$$f(\boldsymbol{x}_*) \leq f(\boldsymbol{x}) \qquad \text{for all } \boldsymbol{x} \in \mathcal{N} \cap \Omega.$$

2. $\boldsymbol{x}_*$ is a **strict** local solution of the OP if:

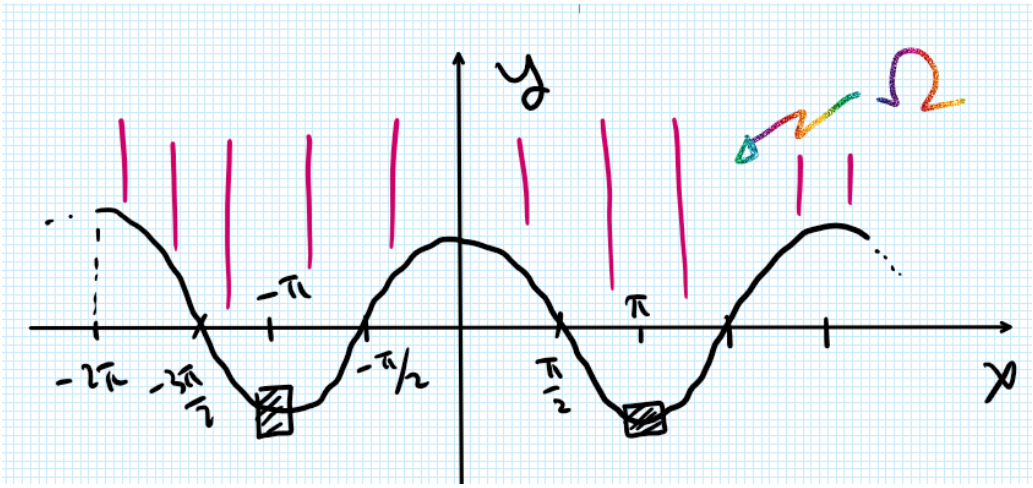


Figure 12.1: The feasible set for the OP (12.5)

- $\boldsymbol{x}_* \in \Omega$,
- There exists a neighbourhood $\boldsymbol{x}_* \ni \mathcal{N}$ such that:

$$f(\boldsymbol{x}_*) < f(\boldsymbol{x}) \qquad \text{for all } \boldsymbol{x} \in \mathcal{N} \cap \Omega \text{ with } \boldsymbol{x} \neq \boldsymbol{x}_*.$$

3. $\boldsymbol{x}_*$ is an **isolated** local solution of the OP if:
   - $\boldsymbol{x}_* \in \Omega$,
   - There exists a neighbourhood $\boldsymbol{x}_* \ni \mathcal{N}$ such that $\boldsymbol{x}_*$ is the only local solution in $\mathcal{N} \cap \Omega$.

12.1.2 Smoothness

The cost function $f(\boldsymbol{x})$ should be a smooth function as otherwise we can't compute ∇f etc. But the constraint function can have 'kinks', so long as the boundary of the feasible set can be described by a surface that is continuous and also, made up of parts that are smooth ('continuous and piecewise smooth').

Example: Consider an OP where the feasible set is:

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^2 | \|\boldsymbol{x}\|_1 \leq 1\}.$$

Hence,

$$|x| + |y| \leq 1.$$

The boundary of the region Ω is $|x| + |y| = 1$, this can be broken up into four curves:

- First Quadrant: x and y both positive, hence $x + y = 1$, hence $y = 1 - x$.
- Second Quadrant: $x < 0$ and $y > 0$, hence $-x + y = 1$, hence $y = 1 + x$.

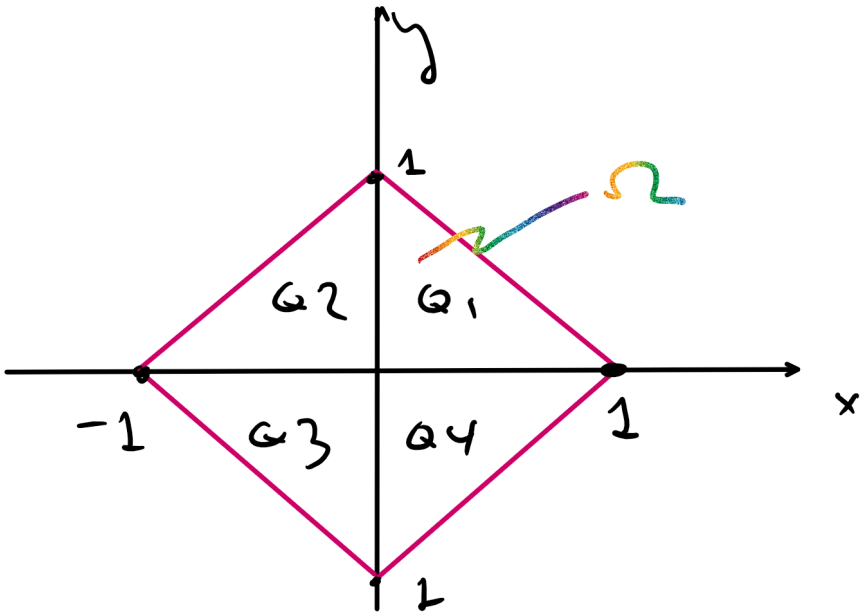


Figure 12.2: The feasible region $|x| + |y| \leq 1$

- Third Quadrant: $y = -1 - x$.
- Fourth Quadrant: $y = -1 + x$.

The feasible region is shown in Figure 12.2.

Non-smooth **unconstrained** optimization problems can often be recast as a smooth constrained OP, e.g. Figure 12.3.

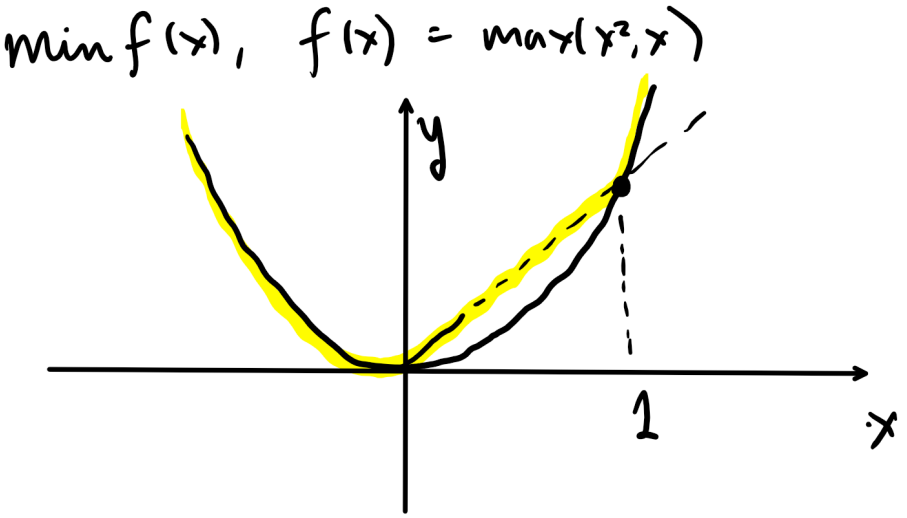


Figure 12.3: Example of an OP with a non-smooth cost function

From the figure, it is clear that there is a minimum at $x_* = 0$. However, for $|x| < 1$, we have:

$$f'(x) = \begin{cases} 2x, & x < 0, \\ 1, & x > 0. \end{cases}$$

Thus, $f'(x)$ has a jump discontinuity at $x_* = 0$. But consider instead the region

$$\Omega = \{(x, y) | y \geq x,\, y \geq x^2\},$$

shown in Figure 12.4. By inspection of the figure,

$$0 = x_* = \min_{\boldsymbol{x} \in \mathbb{R}^2} y, \qquad \text{subject to } \boldsymbol{x} \in \Omega.$$

The cost function is now $f(\boldsymbol{x}) = y$, which is smooth, and the boundary of Ω is a continuous, piecewise differentiable curve.

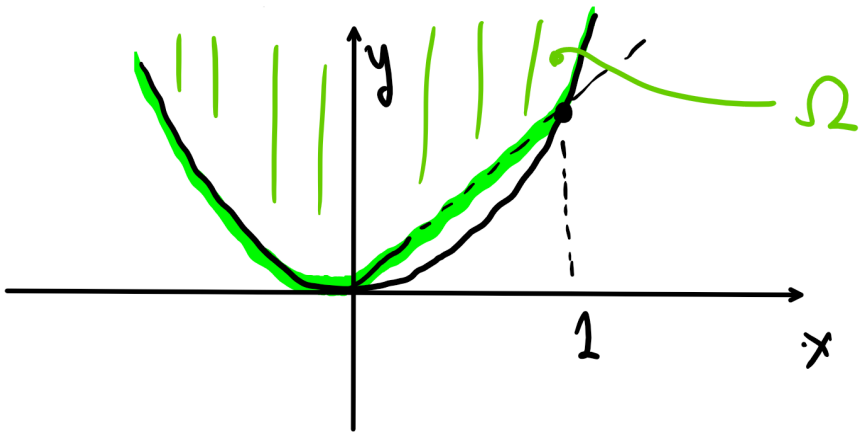


Figure 12.4: Converting the unconstrained OP with a non-smooth cost function to a constrained OP with a smooth cost function

This trick (replacing a non-smooth unconstrained OP with an equivalent smooth, constrained OP) is often employed in more complicated unconstrained OPs where the cost function is non-smooth. For instance, the trick (or a variation on the trick) can be used to reformulate

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \|\boldsymbol{x}\|_1, \qquad \min_{\boldsymbol{x} \in \mathbb{R}^n} \|\boldsymbol{x}\|_\infty$$

as smooth, constrained OPs.

12.1.3 Key Definition – the Active Set

Definition 12.1 *The active set $\mathcal{A}(\boldsymbol{x})$ at any feasible point $\boldsymbol{x} \in \Omega$ consists of:*

- *The indices i of the equality constraints;*
- *The indices i of those inequality constraints satisfying $c_i(\boldsymbol{x}) = 0$.*

In symbols,

$$\mathcal{A}(\boldsymbol{x}) = \mathcal{E} \cup \{i \in \mathcal{I} | c_i(\boldsymbol{x}) = 0\}.$$

Furthermore, we say that at a feasible point $\boldsymbol{x} \in \Omega$, the inequality constraint $c_i(\boldsymbol{x})$ (with $i \in \mathcal{I}$) is **active** if $c_i(\boldsymbol{x}) = 0$; the constraint is **inactive** if $c_i(\boldsymbol{x}) > 0$.

12.2 Worked Example – a single equality constraint

Consider the OP

$$\min f(\boldsymbol{x}) = x + y, \tag{12.6a}$$

subject to

$$c_1(\boldsymbol{x}) = 0, \text{ where } c_1(\boldsymbol{x}) = 2 - x^2 - y^2. \tag{12.6b}$$

Hence, $\mathcal{I} = \emptyset$ and $\mathcal{E} = \{1\}$.

We first of all compute a solution by direction computation as this will be a reference point. The solution can be found by going over to polar coordinates, noting that $x^2 + y^2 = 1$, hence

$$x = \sqrt{2}\cos\theta, \qquad y = \sqrt{2}\sin\theta, \qquad 0 \leq \theta < 2\pi.$$

Thus, $f(\boldsymbol{x}) = \widetilde{f}(\theta) = \sqrt{2}(\cos\theta + \sin\theta)$. To compute the minimum, we solve for θ in

$$\frac{\mathrm{d}\widetilde{f}}{\mathrm{d}\theta} = 0,$$

hence $-\sin\theta + \cos\theta = 0$, hence $\tan\theta = 1$, hence $\theta = \pi/4$ (Q1) or $\theta = 5\pi/4$ (Q3). We check:

- $\widetilde{f}(\theta = \pi/4) = \sqrt{2}\left(\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}\right) = 2$... MAX
- $\widetilde{f}(\theta = 5\pi/4) = -2$... MIN

Hence,

$$\boldsymbol{x}_* = (-1, -1).$$

Consider also,

$$\begin{aligned} \nabla f(\boldsymbol{x}) &= (1,1), \\ \nabla c_1(\boldsymbol{x}) &= (-2x,-2y) \end{aligned}$$

(we momentarily go over to the row-vector representation of vectors). Hence,

$$\begin{aligned} \nabla f(\boldsymbol{x}_*) &= (1,1), \\ \nabla c_1(\boldsymbol{x}_*) &= (2,2). \end{aligned}$$

So $\nabla f \parallel \nabla c_1$ at $\boldsymbol{x} = \boldsymbol{x}_*$. Hence, at $\boldsymbol{x} = \boldsymbol{x}_*$, there exists a constant λ_1^* such that:

$$\nabla f(\boldsymbol{x}_*) = \lambda_1^* \nabla c_1(\boldsymbol{x}_*), \tag{12.7}$$

and in this specific example, $\lambda_1^* = 1/2$. We now present a general derivation of Equation (12.7).

12.3 A General Derivation

We present a general derivation of the result (12.7), valid for any smooth cost function $f(\boldsymbol{x})$ and also for a single smooth equality constraint $c_1(\boldsymbol{x})$.

The idea is to take any feasible point $\boldsymbol{x} \in \Omega$ such that $c_1(\boldsymbol{x}) = 0$. We then move to a neighbouring feasible point $\boldsymbol{x} + \boldsymbol{\delta}$, where $\boldsymbol{\delta}$ is small:

$$\cancel{c_1(\boldsymbol{x})} + \langle \boldsymbol{\delta}, \nabla c_1(\boldsymbol{x}) \rangle = 0 \qquad \text{(Taylor Expansion)}.$$

Thus,

$$\langle \boldsymbol{\delta}, \nabla c_1(\boldsymbol{x}) \rangle = 0. \tag{12.8}$$

In fact, it is better to use traditional dot-product notation here because we will be relying on geometric intuition in what follows:

$$\boldsymbol{\delta} \cdot \nabla c_1(\boldsymbol{x}) = 0.$$

We now look at the case where $\boldsymbol{x} = \boldsymbol{x}_*$. Then, in a neighbourhood if $\boldsymbol{x}_*$:

$$f(\boldsymbol{x}_* + \boldsymbol{\delta}) \geq f(\boldsymbol{x}_*).$$

This implies:

$$\boldsymbol{\delta} \cdot \nabla f(\boldsymbol{x}_*) = 0.$$

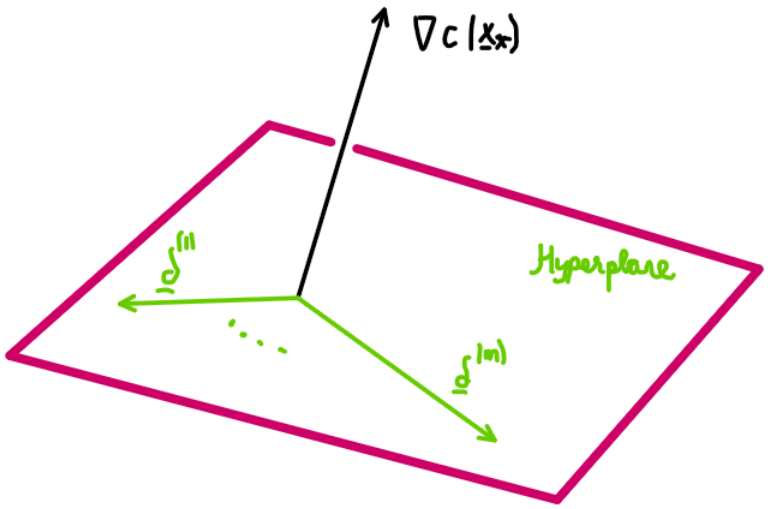


Figure 12.5:

Since $\pm\boldsymbol{\delta}$ is feasible, for the inequality to hold, we must have

$$\boldsymbol{\delta} \cdot \nabla f(\boldsymbol{x}_*) = 0.$$

Note that this is different from unconstrained optimization, where we have $\nabla f(\boldsymbol{x}) = 0$ at $\boldsymbol{x}_*$; here we can only approach the minimum – and hence compute gradients of f – in **feasible** directions.

Refer now to Figure 12.5. The vector $\boldsymbol{\delta}$ must be in a hyperplane orthogonal to $\nabla c_1(\boldsymbol{x}_*)$. The hyperplane is $m = (n-1)$-dimensional, and hence, spanned by m linearly independent vectors $\boldsymbol{\delta}^{(1)}, \cdots, \boldsymbol{\delta}^{(m)}$. For each of these basis vectors we have $\boldsymbol{\delta}^{(j)} \cdot \nabla f(\boldsymbol{x}_*) = 0$. Hence, the only direction in which $\nabla f(\boldsymbol{x}_*)$ can point is back in the direction of $\nabla c_1(\boldsymbol{x}_*)$. Hence, there exists a scalar $\lambda_1^* \in \mathbb{R}$ such that:

$$\nabla f(\boldsymbol{x}_*) = \lambda_1^* \nabla c_1(\boldsymbol{x}_*), \tag{12.9}$$

Notice however that Equation (12.9) holds not only at a local minimum (as desired) but also, at a local maximum.

12.4 The projection operator

Note that $\boldsymbol{\delta}$ is like a 'search direction' in an SD algorithm – albeit now that there are constraints on which direction we can search in. Here, we introduce a theoretical method for working out allowed search directions $\widehat{\boldsymbol{n}} = \boldsymbol{\delta}/\|\boldsymbol{\delta}\|_2$. For this purpose, we introduce a projection operator. This description is valid only when the point of interest $\boldsymbol{x}$ is not a local minimizer or maximizer, since otherwise there is no direction in which further decrease of the cost function is possible.

As such, we introduce the operator

$$\mathbb{P} = \mathbb{I} - \frac{\nabla c_1 \otimes \nabla c_1}{\|\nabla c_1\|_2^2}$$

It can be verified that $\mathbb{P}$ projects a vector $\boldsymbol{v} \in \mathbb{R}^2$ on to a direction perpendicular to ∇c_1:

$$\begin{aligned}
\langle \mathbb{P}\boldsymbol{v}, \nabla c_1 \rangle &= \sum_{i,j} \mathbb{P}_{ij} v_j \frac{\partial c_1}{\partial x_i}, \\
&= \sum_{ij} \left(\delta_{ij} - \widehat{m}_i \widehat{m}_j\right) v_j \widehat{m}_i \|\nabla c_1\|_2, \\
&= \left[\boldsymbol{v} \cdot \widehat{\boldsymbol{m}} - \boldsymbol{v} \cdot \widehat{\boldsymbol{m}} \left(\widehat{\boldsymbol{m}} \cdot \widehat{\boldsymbol{m}}\right)\right] |\nabla c_1\|_2, \\
&= 0.
\end{aligned}$$

Here, we have used the unit vector

$$\widehat{\boldsymbol{m}} = \frac{\nabla c_1}{\|\nabla c_1\|_2},$$

and have used the old-fashioned dot-product notation:

$$\langle \boldsymbol{v}, \widehat{\boldsymbol{m}} \rangle \equiv \boldsymbol{v} \cdot \boldsymbol{m} = \sum_i v_i \widehat{m}_i.$$

This calculation therefore shows that $\mathbb{P}\boldsymbol{v} \perp \nabla c_1$.

As such, we further investigate the search direction $\widehat{\boldsymbol{n}}$, given by:

$$\widehat{\boldsymbol{n}} = \frac{\boldsymbol{s}}{\|\boldsymbol{s}\|_2}, \qquad \boldsymbol{s} = -\mathbb{P}\nabla f(\boldsymbol{x}).$$

We have:

- $\widehat{\boldsymbol{n}} \cdot \nabla c_1 = 0$, by construction.
- Furthermore,

$$\begin{aligned}
\widehat{\boldsymbol{n}} \cdot \nabla f &\propto -\langle \mathbb{P}\nabla f, \nabla f \rangle, \\
&= -\sum_{ij} \mathbb{P}_{ij} g_i g_j, \\
&= -\sum_{ij} \left(\delta_{ij} - \widehat{m}_i \widehat{m}_j\right) g_i g_j, \\
&= -\left[\|\boldsymbol{g}\|_2^2 - \left(\boldsymbol{g} \cdot \widehat{\boldsymbol{n}}\right)^2\right], \\
&= -\|\boldsymbol{g}\|_2^2 \left(1 - \cos^2\theta\right), \\
&\leq 0.
\end{aligned}$$

12.5 The Lagrangian

We can make these calculations more precise by introducing the Lagrangian function,

$$\mathcal{L}(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) - \lambda_1 c_1(\boldsymbol{x}).$$

We note that $\nabla_x \mathcal{L} = \nabla f - \lambda_1 \nabla c_1$. A necessary condition for $\boldsymbol{x}_*$ to be a local minimum of the general equality constraint problem

$$\min f(\boldsymbol{x}), \qquad \text{subject to } c_1(\boldsymbol{x}) = 0$$

is thus that there exists a scalar λ_1^* such that:

$$\nabla_x f(\boldsymbol{x}_*) = \lambda_1^* \nabla c_1(\boldsymbol{x}_*), \tag{12.10a}$$

$$c_1(\boldsymbol{x}_*) = 0. \tag{12.10b}$$

Remark: Equation (13.6) is necessary for a minimum (i.e. if $\boldsymbol{x}_*$ is a minimum, then Equation (13.6) holds). But it is not sufficient. For example, in the OP (12.6), Equation (13.6) is satisfied at the minimum:

$$\text{MIN: } \boldsymbol{x}_* = (-1, -1)^T, \qquad \lambda_1^* = 1/2,$$

but also at the maximum:

$$\text{MAX: } \boldsymbol{x}_* = (1, 1)^T, \qquad \lambda_1^* = -1/2.$$

Obviously, we require necessary and sufficient conditions to nail down a minimum (like for unconstrained optimization).

Chapter 13

Constrained Optimization: Inequality Constraints

Overview

In this Chapter we look at some examples involving inequality constraints, this will help us to 'guess' how to set up a general constrained optimization problem using Lagrange Multipliers, in the case of inequality constraints.

13.1 A Single Inequality Constraint

We look at an example involving a single inequality constraint:

$$\min f(\boldsymbol{x}) = x + y, \tag{13.1a}$$

subject to

$$c_1(\boldsymbol{x}) \geq 0, \qquad c_1(\boldsymbol{x}) = 2 - x^2 - y^2. \tag{13.1b}$$

The solution is still clearly $\boldsymbol{x}_* = (-1, -1)$, which occurs on the constraint boundary $c_1(\boldsymbol{x}) = 0$. Thus, the inequality constraint (13.1b) is active at the minimizer.

13.2 Feasible Descent Directions – General Description

We consider a feasible point

$$\boldsymbol{x} \in \Omega = \{\boldsymbol{x} \in \mathbb{R}^2 | c_1(\boldsymbol{x}) \geq 0\}.$$

We look at a neighbouring feasible point $\boldsymbol{x} \to \boldsymbol{x} + \boldsymbol{\delta}$, we then derive the conditions on $\boldsymbol{\delta}$ such that $\boldsymbol{x} + \boldsymbol{\delta}$ remains feasible. Clearly, we require:

$$c_1(\boldsymbol{x} + \boldsymbol{\delta}) \geq 0.$$

By Taylor expansion, we have, for $\boldsymbol{\delta}$ 'small',

$$c_1(\boldsymbol{x}) + \langle \boldsymbol{\delta}, \nabla c_1(\boldsymbol{x}) \rangle \geq 0, \qquad \|\boldsymbol{\delta}\|_2 \text{ 'small'}. \tag{13.2}$$

Now, however, we can't just set $c_1(\boldsymbol{x}) = 0$ and conclude that $\langle \boldsymbol{\delta}, \nabla c_1(\boldsymbol{x}) \rangle \geq 0$. Instead, we have to look at two cases. When doing this, it will be helpful to switch to the familiar dot-product notation for $\langle \boldsymbol{\delta}, \nabla c_1(\boldsymbol{x}) \rangle$ as this is based on geometric intuition rather than duality.

13.2.1 Case 1 – The minimizer is in the interior

In this case $\boldsymbol{x}_*$ is in the interior of Ω, hence $c_1(\boldsymbol{x}_*) > 0$, and we require $c_1(\boldsymbol{x}_*) + \boldsymbol{\delta} \cdot \nabla c_1(\boldsymbol{x}_*) \geq 0$. This holds for any $\boldsymbol{\delta}$, provided $\|\boldsymbol{\delta}\|_2$ is made sufficiently small. Furthermore, at $\boldsymbol{x} = \boldsymbol{x}_*$, by first-order optimality, we have:

$$\begin{aligned} & f(\boldsymbol{x}_* + \boldsymbol{\delta}) \;\geq\; f(\boldsymbol{x}_*) \\ \implies \boldsymbol{\delta} \cdot \nabla f(\boldsymbol{x}_*) + \tfrac{1}{2} \sum_{i,j} \delta_i \delta_j \left(\frac{\partial^2 f}{\partial x_i \partial x_j} \right)_{\boldsymbol{x}_*} + \cdots \;&\geq\; 0. \end{aligned}$$

Hence, for $\|\boldsymbol{\delta}\|_2$ sufficiently small,

$$\boldsymbol{\delta} \cdot \nabla f(\boldsymbol{x}) \geq 0. \tag{13.3}$$

However, since $\boldsymbol{\delta}$ is now arbitrary, we can take $\boldsymbol{\delta} \propto -\nabla f(\boldsymbol{x}_*)$, which gives $\|\nabla f(\boldsymbol{x}_*)\|^2 \leq 0$, hence:

$$\nabla f(\boldsymbol{x}_*) = 0, \qquad \text{Case 1}.$$

This case is virtually the same as unconstrained optimization.

13.2.2 Case 2 – The minimizer is on the boundary

In this case, $\boldsymbol{x}_*$ is on the constraint boundary, $\boldsymbol{x}_* \in \partial\Omega$, such that $c_1(\boldsymbol{x}_*) = 0$. Referring back to Equation (13.2), we require:

$$\boldsymbol{\delta} \cdot \nabla c_1(\boldsymbol{x}_*) \geq 0.$$

Thus, the set of all allowed vectors $\boldsymbol{\delta}$ is a half-space:

$$\mathcal{H} = \{\boldsymbol{\delta} \in \mathbb{R}^n | \boldsymbol{\delta} \cdot \nabla c_1(\boldsymbol{x}_*) \geq 0\}$$

(the angle between fixed $\nabla c_1(\boldsymbol{x})$ and $\boldsymbol{\delta}$ needs to be less than or equal to $90°$). Vectors in the space can be decomposed as:

$$\boldsymbol{\delta} = \sum_{i=1}^{n-1} \alpha_i \boldsymbol{\delta}^{(i)} + \beta \nabla c(\boldsymbol{x}_*), \qquad \alpha_i \in \mathbb{R}, \qquad \beta \in \mathbb{R}^+ \cup \{0\}.$$

Here, the $\boldsymbol{\delta}^{(i)}$'s span the hyperplane $\boldsymbol{\delta} \cdot \nabla c_1(\boldsymbol{x}_*) = 0$; the notation is the same as Chapter 12. At the local minimum, we have

$$f(\boldsymbol{x}_* + \boldsymbol{\delta}) \geq f(\boldsymbol{x}_*),$$

for all feasible directions $\boldsymbol{\delta} \in \mathcal{H}$. Thus:

$$\boldsymbol{\delta} \cdot \nabla f(\boldsymbol{x}_*) \geq 0, \qquad \boldsymbol{\delta} \in \mathcal{H}$$

(*cf.* Equation (13.3)). In particular:

$$\begin{aligned} \alpha_i \boldsymbol{\delta}^{(i)} \cdot \nabla f(\boldsymbol{x}_*) &\geq 0 \quad \text{, for all } \alpha_i \in \mathbb{R}, \\ \beta \nabla c(\boldsymbol{x}_*) \cdot \nabla f(\boldsymbol{x}_*) &\geq 0 \quad \text{, for all } \beta \in \mathbb{R}^+ \cup \{0\}. \end{aligned}$$

In the first set of conditions, we can take $\alpha_i \to \pm|\alpha_i|$, which gives

$$\boldsymbol{\delta}^{(i)} \cdot \nabla f(\boldsymbol{x}_*) \geq 0 \text{ and } \boldsymbol{\delta}^{(i)} \cdot \nabla f(\boldsymbol{x}_*) \leq 0,$$

hence $\boldsymbol{\delta}^{(i)} \cdot \nabla f(\boldsymbol{x}_*) = 0$. This forces $\nabla f(\boldsymbol{x}_*) \propto \nabla c_1(\boldsymbol{x}_*)$. The second condition further requires the existence of a non-negative scalar λ_1^* such that:

$$\nabla f(\boldsymbol{x}_*) = \lambda_1^* \nabla c_1(\boldsymbol{x}_*). \tag{13.4}$$

Caution: Unlike in equality-constrained problems, the solution of the constrained problem requires a particular sign on λ_1^*, $\lambda_1^* \geq 0$.

13.2.3 Cases 1 and 2 combined

Cases 1 and 2 can be combined together to give:

$$\text{Case 1: } \begin{cases} \nabla f(\boldsymbol{x}_*) & = 0, \\ c_1(\boldsymbol{x}_*) & > 0. \end{cases}$$

$$\text{Case 2: } \begin{cases} \nabla f(\boldsymbol{x}_*) & = \lambda_1^* \nabla c_1, \\ c_1(\boldsymbol{x}_*) & = 0,\ \lambda_1^* \geq 0. \end{cases}$$

Indeed, we can introduce the Lagrangian

$$\mathcal{L}(\boldsymbol{x}, \lambda_1) = f(\boldsymbol{x}) - \lambda_1 c_1(\boldsymbol{x}).$$

Hence, at the local minimum $\boldsymbol{x} = \boldsymbol{x}_*$, there exists a constant $\lambda_1^* \geq 0$ such that:

$$\nabla_x \mathcal{L}(\boldsymbol{x}_*, \lambda_1^*) = 0, \tag{13.5}$$

$$\lambda_1^* c_1(\boldsymbol{x}_*) = 0. \tag{13.6}$$

The condition $\lambda_1^* c_1(\boldsymbol{x}_*) = 0$ is called the **complementarity condition**. It is a way of merging Cases 1 and 2.

Remark: The constraint $c_1(\boldsymbol{x})$ is active when $\lambda_1^* > 0$, then we require $c_1(\boldsymbol{x}_*) = 0$ to satisfy the complementarity condition.

13.3 Two Inequality Constraints

We look at the following OP:

$$\min f(\boldsymbol{x}) = x + y, \tag{13.7a}$$

subject to

$$c_1(\boldsymbol{x}) \geq 0, \qquad c_1(\boldsymbol{x}) = 2 - x^2 - y^2. \tag{13.7b}$$

and

$$c_2(\boldsymbol{x}) \geq 0, \qquad c_2(\boldsymbol{x}) = y. \tag{13.7c}$$

The feasible set here is a half-disc, and the minimum can be worked out by direct computation: the minimum is on the boundary of the feasible set:

$$\boldsymbol{x}_* = (-\sqrt{2}, 0)^T.$$

We now show theoretically why $\boldsymbol{x}_*$ is a local critical point (max or min).

Our approach here is a little bit different from before. We don't make any assumptions about the point $(-\sqrt{2}, 0)^T$, so we re-label it as:

$$\boldsymbol{x}_0 = (-\sqrt{2}, 0)^T.$$

We will go out from $\boldsymbol{x}_0$ in feasible directions $\boldsymbol{d}$ (note the new notation), and in doing so, we will try to reduce the cost function, such that $f(\boldsymbol{x}_0 + \boldsymbol{x}) \leq f(\boldsymbol{x}_0)$. We will end up showing that no such feasible directions exist, meaning the cost function can't be reduced any further and hence, $\boldsymbol{x}_0$ is a local minimum (more precisely, all we can say from this calculation is that $\boldsymbol{x}_0$ is a critical point).

We begin our calculations by noticing that $\boldsymbol{x}_0$ is on the boundary of the feasible region where both constraints are active. This would be 'Case 2' as considered in Section 13.2. We have:

$$\nabla c_1 = (-2x, -2y)^T \overset{\text{Notation}}{=} -2x\mathbf{i} - 2y\mathbf{j}.$$

We look for feasible search directions $\boldsymbol{d} = d_1\mathbf{i} + d_2\mathbf{j}$, hence

$$\boldsymbol{d} \cdot \nabla c_1 = -2xd_1 - 2yd_2.$$

At $\boldsymbol{x}_0 = (-\sqrt{2}, 0)$, we have:

$$\underbrace{\boldsymbol{d} \cdot \nabla c_1(\boldsymbol{x}_0)}_{\geq 0} = 2\sqrt{2}d_1.$$

Hence, feasible search directions satisfy $d_1 \geq 0$. We also have:

$$\nabla c_2 = \mathbf{j}.$$

For feasible search directions, we require $\boldsymbol{d} \cdot \nabla c_2 \geq 0$, hence $d_2 \geq 0$. Thus, we require:

$$d_1 \geq 0, \qquad d_2 \geq 0.$$

We now try to fix $\boldsymbol{d}$ such that going out from $f(\boldsymbol{x}_0)$ to $f(\boldsymbol{x}_0 + \boldsymbol{d})$ reduces the cost function. This requires $\boldsymbol{d} \cdot \nabla f(\boldsymbol{x}_0) \leq 0$. At $\boldsymbol{x}_0$, we have:

$$\begin{aligned} \boldsymbol{d} \cdot \nabla f(\boldsymbol{x}_0) &= (d_1\mathbf{i} + d_2\mathbf{j}) \cdot (\mathbf{i} + \mathbf{j}), \\ &= d_1 + d_2 \end{aligned}$$

Summarizing, we require:

$$d_1 \geq 0, \qquad d_2 \geq 0, \qquad d_1 + d_2 \leq 0,$$

The only way to satisfy all three constraints is to take $\boldsymbol{d} = 0$. In other words, there is no vector $\boldsymbol{d}$ that satisfies $f(\boldsymbol{x}_0 + \boldsymbol{d}) < f(\boldsymbol{x}_0)$. This proves that $\boldsymbol{x}_0$ really is a critical point and the notation $\boldsymbol{x}_* = (-\sqrt{2}, 0)^T$ is justified.

We verify that this is indeed a critical point by finding scalars $\lambda_1^* \geq 0$ and $\lambda_2^* \geq 0$, such that:

$$\nabla_x \mathcal{L}(\boldsymbol{x}_*, \lambda_1^*, \lambda_2^*) = 0, \tag{13.8}$$

$$\lambda_1^* c_1(\boldsymbol{x}_*) = 0, \tag{13.9}$$

$$\lambda_2^* c_2(\boldsymbol{x}) = 0, \tag{13.10}$$

and we further show the reasoning behind why these equations are satisfied at the critical point (actually, the minimum).

13.3.1 Lagrange Multipliers

We now reformulate this problem in terms of Lagrange multipliers. We introduce:

$$\mathcal{L} = f(\boldsymbol{x}) - \lambda_1 c_1(\boldsymbol{x}) - \lambda_2 c_2(\boldsymbol{x}).$$

We seek solutions to

$$\nabla \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = 0,$$

where $\boldsymbol{\lambda}$ is a two-dimensional vector with components λ_1 and λ_2. We denote the solution by $\boldsymbol{x}_*$ and $\boldsymbol{\lambda}^*$, and we further insist:

$$\lambda_i^* \geq 0, \qquad i = 1, 2,$$

as well as specifying the complementarity conditions:

$$\lambda_1^* c_1(\boldsymbol{x}_*) = 0, \qquad \lambda_2^* c_2(\boldsymbol{x}_*) = 0.$$

We compute:

$$\nabla_x \mathcal{L} = (\mathbf{i} + \mathbf{j}) - \lambda_1(-2x\mathbf{i} - 2y\mathbf{j}) - \lambda_2(0\mathbf{i} + \mathbf{j}).$$

At $\nabla_x \mathcal{L} = 0$, we have:

$$1 + 2\lambda_1 x = 0, \tag{13.11a}$$

$$1 + 2\lambda_1 y = \lambda_2. \tag{13.11b}$$

Furthermore, the complementarity conditions read:

$$\lambda_1(2 - x^2 - y^2) = 0, \tag{13.12a}$$
$$\lambda_2 y = 0. \tag{13.12b}$$

Take $(\lambda_2) \times$ (Eq. (13.11)(b)):

$$\lambda_2 + 2\lambda_1 \underbrace{(\lambda_2 y)}_{=0} = \lambda_2^2,$$

hence $\lambda_2 = \lambda_2^2$, and

$$\lambda_2 = 0 \text{ or } \lambda_2 = 1.$$

There are therefore two possibilities to consider.

- Case 1. Take $\lambda_2 = 0$. Then, by Equation (13.11)(b), we have $\lambda_1 y = -1/2$. But $y \geq 0$ in the feasible set, hence $\lambda_1 \leq 0$. This possibility is **rejected**.
- Case 2. Take $\lambda_2 = 1$. Thus, by Equation (13.11)(b), we have $\lambda_1 y = 0$. We require that $\lambda_1 \neq 0$, otherwise Equation (13.11)(a) is inconsistent. Hence, $y = 0$.

So we continue with Case 2. From the active constraint $c_1(\boldsymbol{x}) = 0$, we have $x = \pm\sqrt{2}$. By inspection, we see that $\boldsymbol{x} = (-\sqrt{2}, 0)$ is the minimum, hence:

$$\boldsymbol{x}_* = (-\sqrt{2}, 0),$$

as suspected.

13.4 Summary

The overall point of these exercises is to 'hint' at a kind of first-order optimality condition for inequality constraints:

$$\begin{aligned}
\mathcal{L} &= f(\boldsymbol{x}) - \sum_{i \in \mathcal{I}} \lambda_i c_i(\boldsymbol{x}), \\
\nabla_x \mathcal{L}(\boldsymbol{x}_*, \boldsymbol{\lambda}^*) &= 0, \\
c_i(\boldsymbol{x}_*) &\geq 0, \\
\lambda_i^* &\geq 0, \qquad i \in \mathcal{I}, \\
\lambda_i^* c_i(\boldsymbol{x}_*) &= 0, \qquad i \in \mathcal{I}.
\end{aligned}$$

From the calculations we have done so far, it is easy to see why these conditions should hold. But it is hard to generalize from this to a mixture of equality and inequality constraints. Formulating

conditions such as the above in the case of mixed constraints (equality and inequality) will be the aim of the next few chapters.

Chapter 14

The Tangent Cone and the Set of Linearized Feasible Descent Directions

Overview

In order to derive first-order necessary conditions for optimality in an OP with a mixture of equality and inequality constraints, we need to know about two key concepts called the **Tangent Cone** and the set of Linearized Feasible Descent Directions (LFDDs). It will make our life easier if these are always one and the same, they often are but that is not guaranteed. We therefore derive a condition on the constraints which outlines when these two sets coincide.

14.1 Definition of a Cone in $\mathbb{R}^n$

Definition 14.1 *A set $\mathcal{C}$ is a cone, if for each $\boldsymbol{x} \in \mathcal{C}$, the vector $a\boldsymbol{x}$ is also in $\mathcal{C}$, where a is any positive constant.*

Example: Let $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_m$ be vectors in $\mathbb{R}^n$. Then the set:

$$\mathcal{C} = \{\boldsymbol{a}_1 x_1 + \boldsymbol{a}_2 x_2 + \cdots + \boldsymbol{a}_m x_m | x_i \geq 0,\, i = 1, 2, \cdots, m\}$$

is a cone (See Figure 14.1).

Introduce the matrix

$$A = \begin{array}{c} \uparrow \\ n \\ \downarrow \end{array} \underbrace{\begin{pmatrix} | & | & & | \\ \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_m \\ | & | & & | \end{pmatrix}}_{\leftarrow m \rightarrow} \in \mathbb{R}^{n \times m}$$

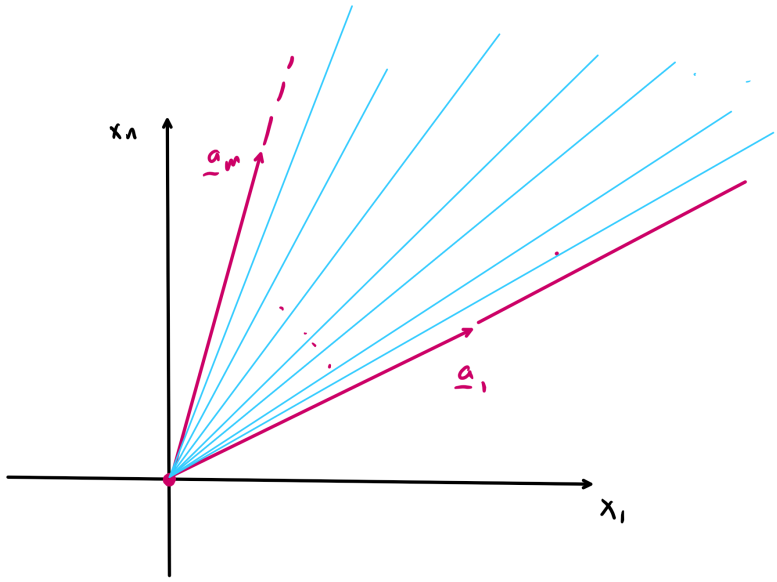


Figure 14.1: A cone in $\mathbb{R}^n$

Thus,

$$\begin{aligned}\mathcal{C} &= \{\boldsymbol{a}_1 x_1 + \boldsymbol{a}_2 x_2 + \cdots + \boldsymbol{a}_m x_m | x_i \geq 0,\ i = 1, 2, \cdots, m\} \\ &= \{A\boldsymbol{x} | \boldsymbol{x} \in \mathbb{R}^m,\ x_i \geq 0,\ i = 1, 2, \cdots, m\}, \\ &:= \mathcal{C}(A).\end{aligned}$$

Check:

$$\begin{aligned}[A\boldsymbol{x}]_i &= \sum_j A_{ij} x_j, \\ &= A_{i1} x_1 + A_{i2} x_2 + \cdots A_{im} x_m.\end{aligned}$$

Hence:

$$A\boldsymbol{x} = \begin{pmatrix} A_{11} \\ A_{21} \\ \vdots \\ A_{n1} \end{pmatrix} x_1 + \begin{pmatrix} A_{12} \\ A_{22} \\ \vdots \\ A_{n2} \end{pmatrix} x_2 + \cdots \begin{pmatrix} A_{1m} \\ A_{2m} \\ \vdots \\ A_{nm} \end{pmatrix} x_m.$$

Hence:

$$A\boldsymbol{x} = \boldsymbol{a}_1 x_1 + \cdots \boldsymbol{a}_m x_m.$$

14.2 The Tangent Cone and the Linearized Feasible Descent Directions

We recall the fundamental OP in constrained optimization:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \text{ subject to } \begin{cases} c_i(\boldsymbol{x}) = 0, & i \in \mathcal{E}, \\ c_i(\boldsymbol{x}) \geq 0, & i \in \mathcal{I}; \end{cases} \tag{14.1}$$

the feasible set is

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n | c_i(\boldsymbol{x}) = 0,\, i \in \mathcal{E},\ c_i(\boldsymbol{x}) \geq 0,\, i \in \mathcal{I}\}. \tag{14.2}$$

Definition 14.2 *The vector $\boldsymbol{d}$ is a tangent vector to Ω at the point $\boldsymbol{x} \in \Omega$ if there is a feasible sequence $\{\boldsymbol{z}_k\}_{k=0}^{\infty}$ approaching $\boldsymbol{x}$ and a sequence of positive scalars $\{t_k\}_{k=0}^{\infty}$ with $t_k \to 0$ as $k \to \infty$, such that:*

$$\boldsymbol{d} = \lim_{k\to\infty} \frac{\boldsymbol{z}_k - \boldsymbol{x}}{t_k}. \tag{14.3}$$

The set of all such tangent vectors at $\boldsymbol{x}$ is called the **tangent cone**, $T_\Omega(\boldsymbol{x})$.

It is straightforward to see that $T_\Omega(\boldsymbol{x})$ is indeed a cone, as per Section 14.1. For take $\boldsymbol{d}$ from Equation (14.3) and multiply by $\alpha > 0$. We have:

$$\begin{aligned} \alpha\boldsymbol{d} &= \lim_{k\to\infty} \frac{\alpha\boldsymbol{z}_k - \alpha\boldsymbol{x}}{t_k}, \\ &= \lim_{k\to\infty} \frac{\boldsymbol{z}_k - \boldsymbol{x}}{\frac{1}{\alpha}t_k}, \end{aligned}$$

from which we identify the feasible sequence $\{\boldsymbol{z}_k\}_{k=0}^{\infty}$, and the sequence of positive scalars $\{\alpha^{-1}t_k\}_{k=0}^{\infty}$ satisfying Definition 14.2. Thus, $\alpha\boldsymbol{d} \in T_\Omega(\boldsymbol{x})$, hence $T_\Omega(\boldsymbol{x})$ is a cone.

Definition 14.3 *The set of* **Linearized Feasible Descent Directions** *at $\boldsymbol{x}$ is given by:*

$$\mathcal{F}(\boldsymbol{x}) = \left\{ \boldsymbol{d} \in \mathbb{R}^n \;\middle|\; \begin{array}{ll} \langle \boldsymbol{d}, \nabla c_i(\boldsymbol{x})\rangle = 0, & i \in \mathcal{E} \\ \langle \boldsymbol{d}, \nabla c_i(\boldsymbol{x})\rangle \geq 0, & i \in \mathcal{I} \cap \mathcal{A}(\boldsymbol{x}_*). \end{array} \right\}.$$

We abbreviate this term as **LFDD**.

Example: In this example we look at a case involving a **single equality constraint**. We look at the OP

$$\min f(\boldsymbol{x}) = x + y, \text{ subject to } c_1(\boldsymbol{x}) = 0,$$

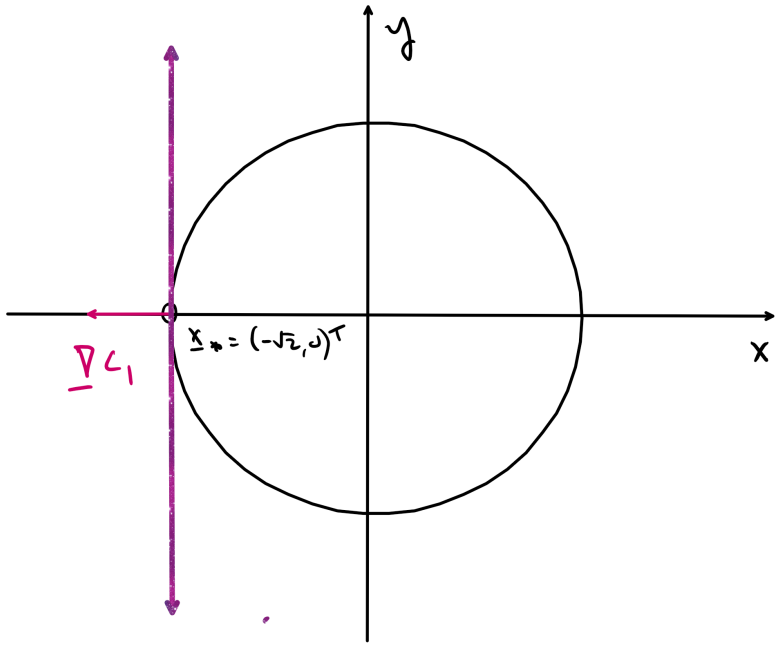


Figure 14.2:

where $c_1(\boldsymbol{x}) = 2 - x^2 - y^2$.

The set of LFDDs at $\boldsymbol{x} = (-\sqrt{2}, 0)^T$ consists of all vectors in the line shown in Figure 14.2. Hence,

$$\mathcal{F}(\boldsymbol{x}) = \{(0, d_2)^T | d_2 \in \mathbb{R}\}.$$

Similarly, to construct feasible sequences which tend to $\boldsymbol{x} = (-\sqrt{2}, 0)^T$, we take:

$$\boldsymbol{x}_k = (\sqrt{2}\cos(\pi - \theta_k), \pm\sqrt{2}\sin(\pi - \theta_k))^T,$$

and we take $\theta_k \to 0$. We have:

$$\begin{aligned} \boldsymbol{d} &= \lim_{\theta_k \to 0} \frac{(\sqrt{2}\cos(\pi - \theta_k) + \sqrt{2}, \pm\sqrt{2}\sin(\pi - \theta_k))^T}{\theta_k}, \\ &= \lim_{\theta_k \to 0} \left(\tfrac{1}{2}\theta_k + O(\theta_k^2), \pm\sqrt{2} + O(\theta_k)\right)^T, \\ &= (0, \pm\sqrt{2})^T. \end{aligned}$$

This exhausts all the possible tangent directions, hence:

$$\begin{aligned} \mathcal{F}_\Omega(\boldsymbol{x}) &= \{\alpha\boldsymbol{y} | \boldsymbol{y} = (0,1)^T \text{ or } \boldsymbol{y} = (0,-1)^T, \alpha > 0\}, \\ &= \{(0, d_2)^T | d_2 \in \mathbb{R}\}. \end{aligned}$$

Thus, in this example,

$$\mathcal{F}_\Omega(\boldsymbol{x}) = T_\Omega(\boldsymbol{x}).$$

Example: If, in the previous example, we replace the constraint function with $c_1(\boldsymbol{x}) = (2-x^2-y^2)^2$, then we have:

$$\begin{aligned}\frac{\partial c_1}{\partial x} &= (2-x^2-y^2)(-2x),\\ \frac{\partial c_1}{\partial y} &= (2-x^2-y^2)(-2y),\end{aligned}$$

hence $\nabla c_1(\boldsymbol{x}) = 0$ at $\boldsymbol{x} = (-\sqrt{2}, 0)^T$ and indeed, $\nabla c_1 = 0$ for all $\boldsymbol{x} \in \Omega$. Thus,

$$\langle \boldsymbol{d}, \nabla c_1 \rangle = \langle \boldsymbol{d}, 0 \rangle = 0,$$

for all $\boldsymbol{d} \in \mathbb{R}^2$, hence,

$$\mathcal{F}_\Omega(\boldsymbol{x}) = \mathbb{R}^2.$$

The tangent cone stays the same, hence for this example, $\mathcal{F}_\Omega(\boldsymbol{x}) \neq T_\Omega(\boldsymbol{x})$. Thus, it is not always the case that $\mathcal{F}_\Omega(\boldsymbol{x}) = T_\Omega(\boldsymbol{x})$.

Example: In this example we look at something based on the previous set of calculations, but involving an **inequality constraint**. Hence, the OP is:

$$\min f(\boldsymbol{x}) = x + y, \text{ subject to } c_1(\boldsymbol{x}) \geq 0, \qquad c_1(\boldsymbol{x}) = 2 - x^2 - y^2.$$

In a previous chapter, we computed the minimum at $\boldsymbol{x}_* = (-1, -1)^T$ and hence, $\min f(\boldsymbol{x}_*) = -2$. Here, we look at the tangent cone and the set of LFDDs at a different point: $\boldsymbol{x} = (-\sqrt{2}, 0)^T$.

To construct the tangent cone, we look for feasible sequences. These can be got by re-arranging Equation (14.3):

$$\boldsymbol{z}_k = \boldsymbol{x} + \boldsymbol{d}t_k + \boldsymbol{\epsilon_k},$$

where $\boldsymbol{\epsilon}_k$ is an error term that goes to zero as $k \to \infty$. Thus, the tail of all possible feasible sequences resemble points along rays; the rays live in the feasible region and point towards $\boldsymbol{x}$. The tangent vectors $\boldsymbol{d}$ are the rays. From Figure 14.3, we see that any $\boldsymbol{d}$ with $d_1 \geq 0$ will do, hence

$$T_\Omega(\boldsymbol{x}) = \{\boldsymbol{d} \in \mathbb{R}^2 | d_1 \geq 0\}.$$

We also look at the set of LFDDs. We require $\langle \boldsymbol{d}, \nabla c_1(\boldsymbol{x}) \rangle \geq 0$ at $\boldsymbol{x} = (-\sqrt{2}, 0)$, hence

$$(d_1\mathbf{i} + d_2\mathbf{j}) \cdot \Big[-2(-\sqrt{2})\mathbf{i} - 2(y=0)\mathbf{j}\Big] \geq 0,$$

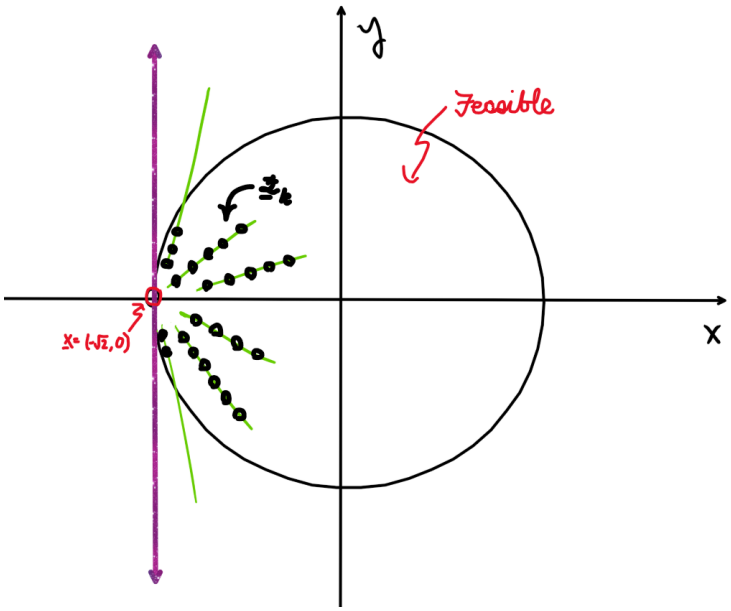


Figure 14.3: Feasible sequences $\boldsymbol{z}_k$ tending to the point of interest $\boldsymbol{x}_* = (-\sqrt{2}, 0)^T$

hence $d_1 \geq 0$, hence

$$\mathcal{F}_\Omega(\boldsymbol{x}) = \{\boldsymbol{d} \in \mathbb{R}^2 | d_1 \geq 0\}.$$

so in this case, $T_\Omega(\boldsymbol{x}) = \mathcal{F}_\Omega(\boldsymbol{x})$.

Example: We look at a rather strange OP where $T_\Omega(\boldsymbol{x}) \neq \mathcal{F}_\Omega(\boldsymbol{x})$. As such, consider:

$$\min f(\boldsymbol{x}) = x + y, \text{ subject to } \begin{cases} c_1(\boldsymbol{x}) \geq 0, & c_1(\boldsymbol{x}) = 1 - x^2 - (y-1)^2, \\ c_2(\boldsymbol{x}) \geq 0, & c_2(\boldsymbol{x}) = -y. \end{cases}$$

We first look at the feasible set in detail:

- Feasible points are in a disc centred at $(0, 1)$ of radius 1;
- Feasible points are in the **lower** half-plane $y \geq 0$.

Hence, the feasible set is precisely one point, $\boldsymbol{x} = (0, 0)^T$ (see Figure 14.4).

We compute the tangent cone at this point. Again, we do this by taking feasible sequences whose tails have the form:

$$\boldsymbol{z}_k = \boldsymbol{x} + \boldsymbol{d}t_k + \boldsymbol{\epsilon_k},$$

where $\boldsymbol{\epsilon}_k$ is an error term that goes to zero as $k \to \infty$. The only way for this to hold in the limit as $t_k \to \infty$ is to take $\boldsymbol{d} = \{(0, 0)^T\}$, thus, the tangent cone is a single point:

$$T_\Omega(\boldsymbol{x}) = \{(0, 0)^T\}.$$

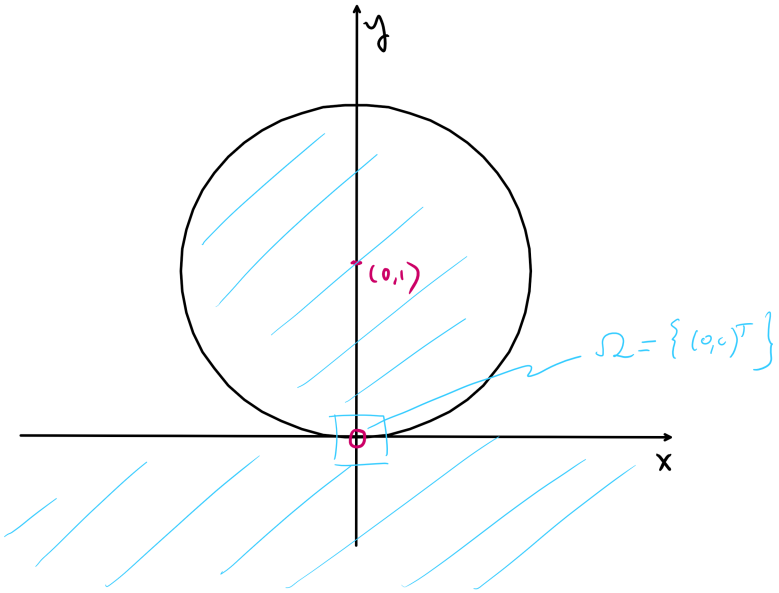


Figure 14.4: Pathological example where the feasible set consists only of a single point

We now look at $\mathcal{F}(\boldsymbol{x})$. From constraint 1 we require:

$$\boldsymbol{d} \cdot \nabla c_1(\boldsymbol{x}) \geq 0,$$

We compute:

$$(d_1\mathbf{i} + d_2\mathbf{j}) \cdot (-2x\mathbf{i} - 2(y-1)\mathbf{j})_{(x=0,y=0)} \geq 0,$$

hence $d_2 \geq 0$.

From constraint 2, we require:

$$\boldsymbol{d} \cdot \nabla c_2(\boldsymbol{x}) \geq 0,$$

We compute:

$$(d_1\mathbf{i} + d_2\mathbf{j}) \cdot (-\mathbf{j}) \geq 0,$$

hence $d_2 \leq 0$.

Putting the conditions from Constraints 1 and 2 together, we require $d_2 = 0$, hence

$$\mathcal{F}_\Omega(\boldsymbol{x}) = \{(d_1, 0)^T | d_1 \in \mathbb{R}\}.$$

But $T_\Omega(\boldsymbol{x})$ is just a single point. Hence,

$$\mathcal{F}(\boldsymbol{x}) \neq T_\Omega(\boldsymbol{x}).$$

14.3 LICQ

In general, we require an extra condition on the constraints for $\mathcal{F}_\Omega(\boldsymbol{x})$ and $T_\Omega(\boldsymbol{x})$ to be the same. On such condition is the Linear Independence Constraint Qualification (LICQ):

Definition 14.4 (LICQ) *Given the point $\boldsymbol{x}$ and the active set $\mathcal{A}(\boldsymbol{x})$, we say that the LICQ holds at $\boldsymbol{x}$ if the active constraints $\nabla c_i(\boldsymbol{x})$, with $i \in \mathcal{A}(\boldsymbol{x})$ are linearly independent.*

Example: From the previous example (Figure 14.4):

$$\begin{aligned} \nabla c_1(0,0) &= (0,2), \\ \nabla c_2(0,0) &= (0,-1). \end{aligned}$$

Thus, ∇c_1 and ∇c_2 are co-linear, so the LICQ does not hold.

Remark: If the LICQ holds, then none of the active constraint gradients can be zero.

Chapter 15

First-Order Necessary Conditions: Background

Overview

We state the necessary conditions for a vector $\boldsymbol{x}_*$ to be the minimizer of the canonical constrained optimization problem.

15.1 Introduction

Having looked through many specific examples of constrained optimization problems, we now formulate some general principles. The aim here is to state the first-order necessary conditions for optimality. We work with the canonical constrained OP:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \text{ subject to } \begin{cases} c_i(\boldsymbol{x}) = 0, & i \in \mathcal{E}, \\ c_i(\boldsymbol{x}) \geq 0, & i \in \mathcal{I}. \end{cases} \tag{15.1}$$

The formulation we develop here is worthy of deep study, as it is the basis of many of the numerical algorithms for constrained optimization. Motivated by the previous examples, we introduce:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \sum_{i\in\mathcal{E}\cup\mathcal{I}} \lambda_i c_i(\boldsymbol{x}) \tag{15.2}$$

We have the following theorem:

Theorem 15.1 *Suppose that $\boldsymbol{x}_*$ is a minimizer of the OP* (15.1)*. Furthermore, suppose that f and the c_i's are continuously differentiable, and that the LICQ holds at $\boldsymbol{x}_*$. Then, there exists a*

vector $\boldsymbol{\lambda}_*$ with components λ_i^*, with $i \in \mathcal{E} \cup \mathcal{I}$, such that the following conditions hold:

$$\begin{aligned}
\nabla_x \mathcal{L}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*) &= 0, && && \text{(15.3a)}\\
c_i(\boldsymbol{x}_*) &= 0, && i \in \mathcal{E}, && \text{(15.3b)}\\
c_i(\boldsymbol{x}_*) &\geq 0, && i \in \mathcal{I}, && \text{(15.3c)}\\
\lambda_i &\geq 0, && i \in \mathcal{I}, && \text{(15.3d)}\\
\lambda_i^* c_i(\boldsymbol{x}_*) &= 0, && i \in \mathcal{I} \cup \mathcal{E}. && \text{(15.3e)}
\end{aligned}$$

Some observations:

- Equations (15.3) are called the **Karush–Kuhn–Tucker** conditions (KKT). In particular, Equation (15.3)(e) is called the complementary condition, this condition imples that either the constraint i is active, or that $\lambda_i^* = 0$, or possibly, both.
- As the Lagrange multipliers corresponding to the inactive constraints are zero, we can re-write Equation (15.3)(a) as:

$$0 = \nabla_x \mathcal{L}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*) = \nabla_x f(\boldsymbol{x}_*) - \sum_{i \in \mathcal{A}(\boldsymbol{x}_*)} \lambda_i^* \nabla c_i(\boldsymbol{x}_*), \tag{15.4}$$

  where the sum here is over active constraints only.

We furthermore have the following definition:

Definition 15.1 (Strict Complementarity) *Given a local solution $\boldsymbol{x}_*$ of the OP* (15.1) *and a vector $\boldsymbol{\lambda}_*$ satisfying the KKT conditions* (15.3)*, we say that strict complementarity holds if exactly one of λ_i^* or $c_i(\boldsymbol{x}_*)$ is zero for each $i \in \mathcal{I}$. In other words, we have $\lambda_i^* > 0$ for $i \in \mathcal{A}(\boldsymbol{x}_*) \cap \mathcal{I}$.*

Some more observations:

- Satisfaction of the strict complementarity conditions often makes it easier for a numerical algorithm to determine the active set and hence, to converge rapidly to $\boldsymbol{x}_*$.
- For a given OP and a solution point $\boldsymbol{x}_*$, there may be many vectors $\boldsymbol{\lambda}_*$ which satisfy the KKT conditions. HOwever, by imposing the LICQ, the vector $\boldsymbol{\lambda}_*$ is unique.

Having stated the first-order necessary conditions for optimality, the aim of the rest of this chapter is twofold:

- Provide yet more examples to help us to understand the KKT conditions.

- Prove some preliminary (but highly nontrivial results) which will enable us to prove the KKT conditions in a later chapter.

As a first step, we will check that the LICQ makes the λ_i^*'s unique.

15.1.1 LICQs

Theorem 15.2 *If the c_i's satisfy the LICQ at $\boldsymbol{x}_*$, then the Lagrange multipliers in the KKT conditions are unique.*

Proof: Suppose that there is a set of Lagrange multipliers λ_i^* (with $i \in \mathcal{E} \cup \mathcal{I}$ satisfying the KKT conditions, and another set μ_i^* with $i \in \mathcal{E} \cup \mathcal{I}$ satisfying the same. We have:

$$0 = \nabla_x \mathcal{L}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*) = \nabla_x f(\boldsymbol{x}_*) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i^* \nabla c_i(\boldsymbol{x}_*) = \nabla_x f(\boldsymbol{x}_*) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \mu_i^* \nabla c_i(\boldsymbol{x}_*).$$

Thus,

$$\sum_{i \in \mathcal{E} \cup \mathcal{I}} \nabla c_i(\boldsymbol{x}_*) \left(\lambda_i^* - \mu_i^*\right) = 0. \tag{15.5}$$

As the gradients $\nabla c_i(\boldsymbol{x}_*)$ are all linearly independent, the only way for Equation (15.5) to be zero is if $\lambda_i^* = \mu_i^*$, for all $i \in \mathcal{E} \cup \mathcal{I}$. This establishes the uniqueness of the λ_i^*'s under the LICQ. ∎

15.1.2 Example

Consider the OP

$$\min f(\boldsymbol{x}) = \left(x - \tfrac{3}{2}\right)^2 + \left(y - \tfrac{1}{2}\right)^2,$$

subject to:

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 1 - x - y \\ 1 - x + y \\ 1 + x - y \\ 1 + x + y \end{bmatrix} \geq 0.$$

Solution: We first of all do a solution by direct computation, this is possible here because the example is relatively simple. The feasible region is shown in Figure 15.1, from the figure it is clear that the minimizer $\boldsymbol{x}_*$ satisfies

$$(x_*, y_*) \in L_1, \qquad L_1 : y = 1 - x.$$

We introduce

$$\begin{aligned}\widetilde{f}(x) &= \left(x-\tfrac{3}{2}\right)^2 + \left[(y=1-x)-\tfrac{1}{2}\right]^2, \\ &= \left(x-\tfrac{3}{2}\right)^2 + \left(x-\tfrac{1}{2}\right)^2, \\ &= 2x^2 - 4x + \tfrac{5}{2}.\end{aligned}$$

We therefore need to minimize $\widetilde{f}(x)$ subject to $x \in [0,1]$. In other words, we seek x_*, where:

$$x_* = \arg\min_{[0,1]} \left[2x^2 - 4x + \tfrac{5}{2}\right].$$

The solution to this problem can be found by inspection: $x_* = 1$. The solution of the OP is thus:

$$\boldsymbol{x}_* = (1,0)^T.$$

From the figure, it is clear that the constraints c_1 and c_2 are both active at this point.

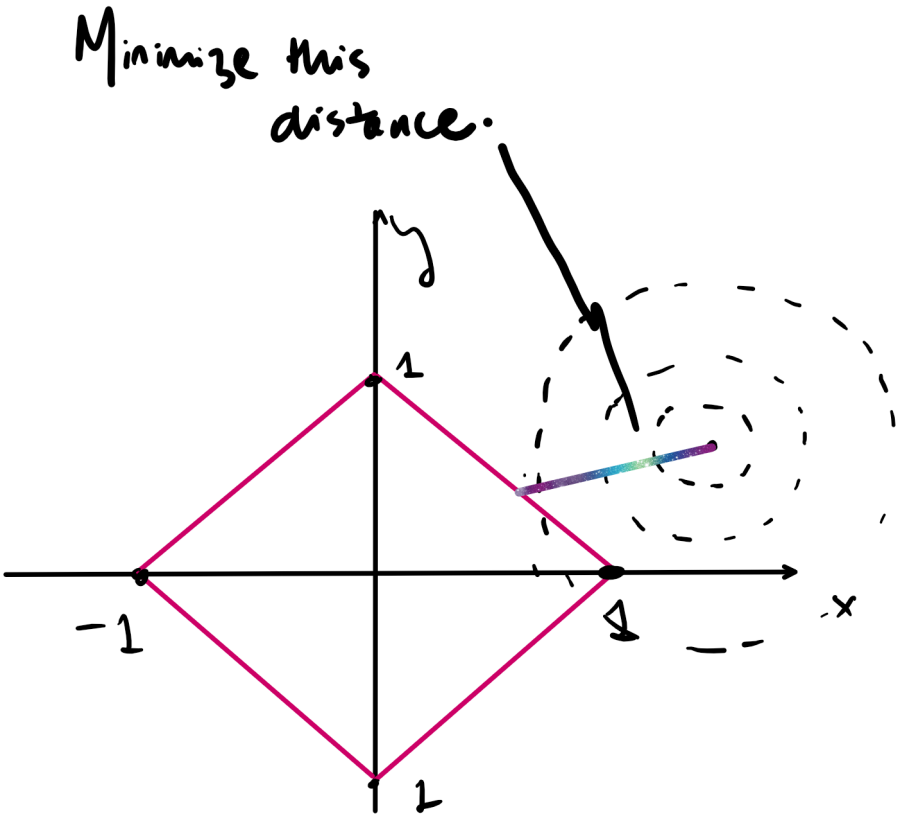


Figure 15.1: Simple example of constrained optimization with validation of the KKT conditions

We now check that the LICQ holds at $\boldsymbol{x}_*$. We have:

$$\nabla f(\boldsymbol{x}_*) = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \qquad \nabla c_1(\boldsymbol{x}_*) = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \qquad \nabla c_2(\boldsymbol{x}_*) = \begin{pmatrix} -1 \\ 1 \end{pmatrix}.$$

We check the LICQ by looking at the matrix formed by the columns of ∇c_1 and ∇c_2:

$$A = \begin{pmatrix} -1 & -1 \\ -1 & 1 \end{pmatrix}.$$

As $\det(A) = -2$, A has full rank, hence, ∇c_1 and ∇c_2 are linearly independent, hence the LICQ holds.

We next compute the Lagrange multipliers for the active constraints. Hence, we set

$$\nabla f(\boldsymbol{x}) - \sum_{i \in \mathcal{A}(\boldsymbol{x}_*)} \lambda_i^* \nabla c_i^(\boldsymbol{x}_*) = 0,$$

hence

$$\begin{pmatrix} -1 \\ -1 \end{pmatrix} - \lambda_1^* \begin{pmatrix} -1 \\ -1 \end{pmatrix} - \lambda_2^* \begin{pmatrix} -1 \\ 1 \end{pmatrix} = 0$$

or

$$\begin{aligned} \lambda_1^* + \lambda_2^* &= 1, \\ \lambda_1^* - \lambda_2^* &= 1, \end{aligned}$$

hence finally, $\lambda_1^* = 1$ and $\lambda_2^* = 0$. Notice that the compatibility conditions are not strictly complementary, since we have:

$$\begin{aligned} c_1(\boldsymbol{x}_*) = 0 \quad , \qquad & \lambda_1^* = 1, \\ c_2(\boldsymbol{x}_*) = 0 \quad , \qquad & \lambda_2^* = 0. \end{aligned}$$

If, instead, the cost function is:

$$f(\boldsymbol{x}) = \left(x - \tfrac{3}{2}\right)^2 + \left(y - \tfrac{1}{2}\right)^4,$$

(note the power on $(y - 1/2)$), with the same constraints as before, the compatibility conditions become strictly complementary. We check this as follows. We have:

$$\nabla f(\boldsymbol{x}) = \left(2(x - \tfrac{3}{2}), 4\left(y - \tfrac{1}{2}\right)^3\right)^T$$

hence

$$\begin{aligned}\nabla f(\boldsymbol{x}_*) &= \left(-1, 4\left(-\tfrac{1}{2}\right)\left(-\tfrac{1}{2}\right)\left(-\tfrac{1}{2}\right)\right)^T, \\ &= \left(-1, \left(-\tfrac{1}{2}\right)\right)^T.\end{aligned}$$

The equation $\nabla f(\boldsymbol{x}_*) - \sum_{i\in\mathcal{A}(\boldsymbol{x}_*)} \lambda_i^* \nabla c_i(\boldsymbol{x}_*) = 0$ now becomes:

$$\begin{pmatrix} -1 \\ -1/2 \end{pmatrix} - \lambda_1^* \begin{pmatrix} -1 \\ -1 \end{pmatrix} - \lambda_2^* \begin{pmatrix} -1 \\ 1 \end{pmatrix} = 0$$

with solution $\lambda_1^* = 1$ and $\lambda_2^* = 1/4$. Thus:

$$\begin{aligned} c_1(\boldsymbol{x}_*) = 0 \quad , \qquad & \lambda_1^* = 1, \\ c_2(\boldsymbol{x}_*) = 0 \quad , \qquad & \lambda_1^* = 1/4. \end{aligned}$$

Hence, the complementarity conditions

$$\lambda_i^* c_i(\boldsymbol{x}_*) = 0, \qquad i \in \mathcal{A}(\boldsymbol{x}_*) \cap \mathcal{I}$$

hold strictly.

15.2 Condition for the Tangent Cone and the set of LFDDs to coincide

We now build up to a proof of the KKT conditions. We are going to show first of all that:

$$T_\Omega(\boldsymbol{x}_*) = \mathcal{F}_\Omega(\boldsymbol{x}_*),$$

provided the LICQ is satisfied at a feasible point $\boldsymbol{x}_*$. We first of all introduce some notation. Recall, $\mathcal{A}(\boldsymbol{x}_*)$ is the active set: if

$$c_{i_1}(\boldsymbol{x}_*) = 0, \qquad c_{i_2}(\boldsymbol{x}_*) = 0, \qquad c_{i_m}(\boldsymbol{x}_*) = 0,$$

are the active constraints, then the active set at $\boldsymbol{x}_*$ is:

$$\mathcal{A}(\boldsymbol{x}_*) = \{i_1, i_2, \cdots, i_m\}.$$

We now introduce the matrix $A \in \mathbb{R}^{m\times n}$:

$$A(\boldsymbol{x}_*) = \begin{array}{c}\uparrow\\ m\\ \downarrow\end{array} \underbrace{\begin{pmatrix} \frac{\partial c_{i_1}}{\partial x_1} & \frac{\partial c_{i_1}}{\partial x_2} & \cdots & \frac{\partial c_{i_1}}{\partial x_n}, \\ \vdots & & & \\ \frac{\partial c_{i_m}}{\partial x_1} & \frac{\partial c_{i_m}}{\partial x_2} & \cdots & \frac{\partial c_{i_m}}{\partial x_n} \end{pmatrix}_{\boldsymbol{x}_*}}_{\leftarrow n\rightarrow}, \qquad A(\boldsymbol{x}_*) \in \mathbb{R}^{m\times n}.$$

We drop the $\boldsymbol{x}_*$ notation in A is the dependence is obvious. Notice also that:

$$m < n,$$

which makes sense: in general, the optimization could be 100-dimensional or 1000-dimensional, but we might have 1-2 constraints.

15.2.1 The kernel of A

We look at the kernel of A:

$$\begin{array}{c}\uparrow\\ m\\ \downarrow\end{array} \underbrace{\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & & \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}}_{\leftarrow n\rightarrow} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} = \begin{pmatrix} a_{11}z_1 + \cdots a_{1n}z_n \\ \vdots \\ a_{m1}z_1 + \cdots a_{mn}z_n \end{pmatrix}$$

Hence:

$$\sum_{i=1}^{n} a_{ji}z_i = 0, \qquad j \in \{1, 2, \cdots, m\}. \tag{15.6}$$

In general:

- $\boldsymbol{z} = (z_1, \cdots, z_n)^T$ is a vector with n variables.
- But there are m constraints (from Equation (15.6)).
- So there are only $n - m$ free variables in the vector $\boldsymbol{z}$.

Hence, the kernel of A is $(n - m)$-dimensional. Let the basis of $\ker(A)$ be $\{\boldsymbol{z}^{(1)}, \cdots, \boldsymbol{z}^{(n-m)}\}$. Form the matrix:

$$Z = \begin{array}{c}\uparrow\\ n\\ \downarrow\end{array} \underbrace{\begin{pmatrix} | & \cdots & | \\ \boldsymbol{z}^{(1)} & \cdots & \boldsymbol{z}^{(n-m)} \\ | & \cdots & | \end{pmatrix}}_{\leftarrow (n-m)\rightarrow} \in \mathbb{R}^{n\times(n-m)}.$$

Hence, $AZ = 0$. We have the following lemma:

Lemma 15.1 *Suppose that the LICQ are satisfied at $\boldsymbol{x}_*$. Then the matrix*

$$\begin{pmatrix} A \\ Z^T \end{pmatrix} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & & \\ a_{m1} & \cdots & a_{mn} \\ z_1^{(1)} & & z_n^{(1)} \\ \cdots & & \\ z_1^{(n-m)} & & z_n^{(n-m)} \end{pmatrix} \in \mathbb{R}^{n\times n} \tag{15.7}$$

has full row rank.

Proof: We start by noticing:

- The first m rows are linearly independent, by the LICQ.
- The last $n-m$ rows are linearly independent, by construction of Z.

So it remains to check that the first m rows are linearly independent of the last $n-m$ rows. Assume for contradiction that they are not. Then we can write (for example):

$$\left(z_1^{(1)}, \cdots, z_n^{(1)}\right)^T = \sum_{j=1}^{m} \mu_j \left(a_{j1}, \cdots a_{jn}\right)^T. \tag{15.8}$$

But

$$\sum_{i=1}^{n} a_{ji} z_i^{(1)} = 0,$$

by definition of $\boldsymbol{z}^{(1)}$ as being in the kernel of A. Combine these two results:

$$\begin{aligned} z_i^{(1)} &\overset{\text{Eq. (15.8)}}{=} \sum_{j=1}^{m} \mu_j a_{ji}, \\ &\overset{\text{re-index}}{=} \sum_{k=1}^{m} \mu_k a_{ki} \end{aligned} \tag{15.9}$$

Since $\boldsymbol{z}^{(1)}$ is in the kernel of A:

$$0 \overset{\text{kernel}}{=} \sum_{i=1}^{n} a_{ji} z_i^{(1)} \overset{\text{Eq. (15.9)}}{=} \sum_{i=1}^{n}\sum_{k=1}^{m} \mu_k a_{ki} a_{ji}.$$

Hence:

$$\sum_{ik} (A)_{ki} (A^T)_{ij} \mu_k = 0,$$

or:

$$AA^T\boldsymbol{\mu} = 0, \qquad \boldsymbol{\mu} = (\mu_1, \cdots, \mu_m)^T.$$

Hence, $\langle \boldsymbol{\mu}, AA^T\boldsymbol{\mu} \rangle = 0$, hence $\langle A^T\boldsymbol{\mu}, A^T\boldsymbol{\mu} \rangle = 0$, hence $A^T\boldsymbol{\mu} = 0$. In index form this is:

$$\mu_1(a_{11}, a_{12}, \cdots, a_{1n}) + \cdots \mu_n(a_{m1}, a_{m2}, \cdots a_{mn}) = 0.$$

But A has full row rank, hence $\mu_1, \mu_2, \cdots, \mu_n = 0$, hence $\boldsymbol{z}^{(1)} = 0$, which is a contradiction, since $\boldsymbol{z}^{(1)}$ is a basis vector. Hence, Equation (15.8) is false. Thus, all of the rows in Equation (15.7) are linearly independent. ∎

15.2.2 The result

We now have the following theorem.

Theorem 15.3 *Let $\boldsymbol{x}_*$ be a feasible point. Then the following statements are true:*

- $T_\Omega(\boldsymbol{x}_*) \subset \mathcal{F}_\Omega(\boldsymbol{x}_*)$,
- *If the LICQ holds, then $T_\Omega(\boldsymbol{x}_*) = \mathcal{F}_\Omega(\boldsymbol{x}_*)$.*

Remark: We use $\boldsymbol{x}_*$ here for any feasible point, it doesn't have to be a minimizer.

We now prove the theorem starting with the first part. We assume without loss of generality that all the constraints are active, this just helps with indexing. Let $\boldsymbol{d} \in T_\Omega(\boldsymbol{x}_*)$. Hence, we seek to show that $\boldsymbol{d} \in \mathcal{F}_\Omega(\boldsymbol{x}_*)$. As such, there exists a sequence $\{\boldsymbol{z}_k\}_{k=0}^\infty$ (with the 'tail' of the sequence in Ω), and a sequence of positive scalars $\{t_k\}_{k=0}^\infty$, with $t_k \to 0$ as $k \to \infty$, such that:

$$\lim_{k\to\infty} \frac{\boldsymbol{z}_k - \boldsymbol{x}_*}{t_k} = \boldsymbol{d}. \tag{15.10}$$

Hence, for k sufficiently large,

$$\boldsymbol{z}_k = \boldsymbol{d}t_k + \boldsymbol{x}_* + \boldsymbol{\epsilon}_k, \tag{15.11}$$

where $\|\boldsymbol{\epsilon}_k\| \to 0$ as $k \to \infty$. Furthermore,

$$\frac{\boldsymbol{z}_k - \boldsymbol{x}_*}{t_k} - \boldsymbol{d} = \frac{\boldsymbol{\epsilon_k}}{t_k} \to 0 \text{ as } k \to \infty.$$

Hence, $\boldsymbol{\epsilon}_k$ is 'little-o of t_k':

$$\lim_{k\to\infty} \frac{\|\boldsymbol{\epsilon_k}\|}{t_k} \to 0,$$

so we can re-write Equation (15.11) as:

$$\boldsymbol{z}_k = \boldsymbol{d}t_k + \boldsymbol{x}_* + o(t_k). \tag{15.12}$$

We first of all take $i \in \mathcal{E}$. We have:

$$0 = \frac{1}{t_k} c_i(\boldsymbol{z}_k), \tag{15.13}$$

since $\boldsymbol{z}_k$ is feasible for k sufficiently large. Compare Equations (15.12) and (15.13):

$$0 = \frac{1}{t_k} \left[c_i(\boldsymbol{x}_*) + t_k \boldsymbol{d} \cdot \nabla c_i(\boldsymbol{x}_*) + o(t_k) \right],$$

But $c_i(\boldsymbol{x}_*)0 =$, hence

$$0 = \boldsymbol{d} \cdot \nabla c_i(\boldsymbol{x}_*) + \frac{o(t_k)}{t_k}.$$

Also, $o(t_k)/t_k \to 0$ as $k \to \infty$, hence:

$$0 = \boldsymbol{d} \cdot \nabla c_i(\boldsymbol{x}_*), \qquad i \in \mathcal{E}. \tag{15.14a}$$

Now take $i \in \mathcal{I}$ and repeat the same calculation:

$$0 \leq \frac{1}{t_k} \left[c_i(\boldsymbol{x}_*) + t_k \boldsymbol{d} \cdot \nabla c_i(\boldsymbol{x}_*) + o(t_k) \right],$$

Since i is an active index, we have $c_i(\boldsymbol{x}_*) = 0$, hence

$$0 \leq \boldsymbol{d} \cdot \nabla c_i(\boldsymbol{x}_*) + \frac{o(t_k)}{t_k}.$$

Take $t_k \to \infty$, hence:

$$0 \leq \boldsymbol{d} \cdot \nabla c_i(\boldsymbol{x}_*), \qquad i \in \mathcal{I}. \tag{15.14b}$$

From Equation (15.14)(a) and (b), we see that $\boldsymbol{d} \in \mathcal{F}_\Omega(\boldsymbol{x}_*)$, hence $T_\Omega(\boldsymbol{x}_*) \subset \mathcal{F}_\Omega(\boldsymbol{x}_*)$.

We now move on to the second part of the proof. We introduce $\boldsymbol{d} \in \mathcal{F}_\Omega(\boldsymbol{x}_*)$. We further introduce a map:

$$\begin{aligned} R : \mathbb{R}^n \times \mathbb{R} &\to \mathbb{R}^n, \\ (\boldsymbol{z}, t) &\mapsto R(\boldsymbol{z}, t), \end{aligned}$$

such that

$$R(\boldsymbol{z}, t) = \begin{pmatrix} c(\boldsymbol{z}) - tA(\boldsymbol{x}_*)\boldsymbol{d} \\ Z^T(\boldsymbol{z} - \boldsymbol{x}_* - t\boldsymbol{d}) \end{pmatrix}.$$

Notice that:

$$R(\boldsymbol{x}_*, 0) = 0, \qquad \nabla_z R(\boldsymbol{x}_*, 0) = \begin{pmatrix} A \\ Z^T \end{pmatrix} \text{ is invertible.}$$

So, by the Implicit Function Theorem, there exists a differentiable path $\boldsymbol{z}(t)$ valid on an interval I containing the point $t = 0$, such that $R(\boldsymbol{z}(t), t) = 0$, for all $t \in I$. As such, $\boldsymbol{z}(t)$ is a curve, points

on which are candidates for constructing a tangent sequence.

Therefore, we pick out a sequence $\boldsymbol{z}_k$ along the curve $\boldsymbol{z}(t)$, and a sequence of positive scalars t_k. We have:

$$\begin{aligned} 0 &= R_i(\boldsymbol{z}_k, t_k), \\ &= \cancel{R_i(\boldsymbol{x}_*, 0)} + \left(\frac{\mathrm{d}R_i}{\mathrm{d}t}\right)_{t=0} t_k + O(t_k^2), \\ &= \sum_j \left(\frac{\partial R_i}{\partial z_j}\right)_{(\boldsymbol{x}_*,0)} \left.\frac{\mathrm{d}z_j}{\mathrm{d}t}\right|_{t=0} t_k + \left(\frac{\partial R_i}{\partial t}\right)_{(\boldsymbol{x}_*,0)} t_k + O(t_k^2), \\ &= \sum_j \left(\frac{\partial R_i}{\partial z_j}\right)_{(\boldsymbol{x}_*,0)} (\boldsymbol{z}_k - \boldsymbol{x}_*)_j + \left(\frac{\partial R_i}{\partial t}\right)_{t=0} t_k + O(t_k^2), \end{aligned}$$

In array form, this is:

$$0 = \begin{pmatrix} A(\boldsymbol{x}_*) \\ Z^T \end{pmatrix} (\boldsymbol{z}_k - \boldsymbol{x}_*) + \begin{pmatrix} -A(\boldsymbol{x}_*)\boldsymbol{d} \\ -Z^T\boldsymbol{d} \end{pmatrix} t_k + O(t_k^2).$$

Hence

$$0 = \begin{pmatrix} A(\boldsymbol{x}_*) \\ Z^T \end{pmatrix} \left(\frac{\boldsymbol{z}_k - \boldsymbol{x}_*}{t_k} - \boldsymbol{d}\right) + O(t_k),$$

hence

$$0 = \underbrace{\begin{pmatrix} A(\boldsymbol{x}_*) \\ Z^T \end{pmatrix}}_{\text{Invertible}} \left(\frac{\boldsymbol{z}_k - \boldsymbol{x}_*}{t_k} - \boldsymbol{d}\right) \text{ as } t_k \to 0.$$

But the matrix here is invertible, hence,

$$\frac{\boldsymbol{z}_k - \boldsymbol{x}_*}{t_k} \to \boldsymbol{d} \text{ as } t_k \to 0.$$

We check the feasibility of the sequence. As $R(\boldsymbol{z}_k, t_k) = 0$, we have:

$$c_i(\boldsymbol{z}_k) = t_k[A(\boldsymbol{x}_*)\boldsymbol{d}]_i, \qquad i \in \mathcal{E} \cup \mathcal{I} = \mathcal{A}(\boldsymbol{x}_*).$$

For equality constraints, we have:

$$c_i(\boldsymbol{z}_k) = t_k[A(\boldsymbol{x}_*)\boldsymbol{d}]_i = t_k\boldsymbol{d} \cdot \nabla c_i(\boldsymbol{x}_*) \overset{\text{LFDD}}{=} 0.$$

hence $c_i(\boldsymbol{z}_k) = 0$. For inequality constraints, we have:

$$c_i(\boldsymbol{z}_k) = t_k[A(\boldsymbol{x}_*)\boldsymbol{d}]_i = t_k\boldsymbol{d} \cdot \nabla c_i \overset{\text{LFDD}}{\geq} 0,$$

hence $c_i(\boldsymbol{z}_k) \geq 0$. Thus, the sequence $\boldsymbol{z}_k$ is feasible. Summarizing, we have started with a given LFDD $\boldsymbol{d}$ and have constructed a sequence of points $\boldsymbol{z}_k$ and a sequence of scalars t_k such that:

- The sequence $\boldsymbol{z}_k$ is feasible;
- The sequences $\boldsymbol{z}_k$ and t_k satisfy:

$$\frac{\boldsymbol{z}_k - \boldsymbol{x}_*}{t_k} \to \boldsymbol{d} \text{ as } t_k \to 0.$$

Hence, $\boldsymbol{d} \in T_\Omega(\boldsymbol{x}_*)$, as required. ∎

Chapter 16

First-Order Necessary Conditions: Proof

Overview

We prove the necessary conditions for a vector $\boldsymbol{x}_*$ to be the minimizer of the canonical constrained optimization problem.

16.1 Introduction

We again work with the canonical constrained OP:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \text{ subject to } \begin{cases} c_i(\boldsymbol{x}) = 0, & i \in \mathcal{E}, \\ c_i(\boldsymbol{x}) \geq 0, & i \in \mathcal{I}. \end{cases} \tag{16.1}$$

The formulation we develop here is worthy of deep study, as it is the basis of many of the numerical algorithms for constrained optimization. Motivated by the previous examples, we introduce:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \sum_{i\in\mathcal{E}\cup\mathcal{I}} \lambda_i c_i(\boldsymbol{x}) \tag{16.2}$$

We have the following theorem (KKT conditions), which was introduced previously in Chapter 15 as Theorem 15.1, but which we re-introduce here with a new label:

Theorem 16.1 *Suppose that $\boldsymbol{x}_*$ is a minimizer of the OP* (16.1)*. Furthermore, suppose that f and the c_i's are continuously differentiable, and that the LICQ holds at $\boldsymbol{x}_*$. Then, there exists a*

vector $\boldsymbol{\lambda}_*$ with components λ_i^*, with $i \in \mathcal{E} \cup \mathcal{I}$, such that the following conditions hold:

$$\begin{aligned} \nabla_x \mathcal{L}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*) &= 0, && \text{(16.3a)} \\ c_i(\boldsymbol{x}_*) &= 0, \qquad i \in \mathcal{E}, && \text{(16.3b)} \\ c_i(\boldsymbol{x}_*) &\geq 0, \qquad i \in \mathcal{I}, && \text{(16.3c)} \\ \lambda_i &\geq 0, \qquad i \in \mathcal{I}, && \text{(16.3d)} \\ \lambda_i^* c_i(\boldsymbol{x}_*) &= 0, \qquad i \in \mathcal{I} \cup \mathcal{E}. && \text{(16.3e)} \end{aligned}$$

We will refer to these conditions as KKT1–5, throughout this chapter, and in the assignments. The aim of this chapter then is to prove KKT1–5 from first principles. To do this, we have to get through some preliminary results.

Remark: Since $c_i(\boldsymbol{x}_*) > 0$ for $i \in \mathcal{I} \backslash \mathcal{A}(\boldsymbol{x}_*)$, KKT5 implies that $\lambda_i^* = 0$ for $i \in \mathcal{I} \backslash \mathcal{A}(\boldsymbol{x}_*)$. Hence, KKT1 can be replaced with:

$$\nabla f(\boldsymbol{x}_*) = \sum_{i \in \mathcal{A}(\boldsymbol{x}_*)} \lambda_i^* \nabla c_i(\boldsymbol{x}_*).$$

16.2 Fundamental Necessary Condition

We prove the following lemma:

Lemma 16.1 (Fundamental Necessary Condition) *If $\boldsymbol{x}_*$ is a local minimizer of the generic constrained OP* (16.1)*, then:*

$$\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}_*) \rangle \geq 0, \qquad \textit{for all } \boldsymbol{d} \in T_\Omega(\boldsymbol{x}_*). \tag{16.4}$$

We produce a proof by contradiction. Suppose there is some $\boldsymbol{d} \in T_\Omega(\boldsymbol{x}_*)$ such that:

$$\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}_*) \rangle < 0. \tag{16.5}$$

Let a corresponding feasible sequence be $\{\boldsymbol{z}_k\}_{k=0}^\infty$, such that:

$$\lim_{k \to \infty} \frac{\boldsymbol{z}_k - \boldsymbol{x}_*}{t_k} = \boldsymbol{d}.$$

We have:

$$f(\boldsymbol{z}_k) = f(\boldsymbol{x}_*) + t_k \underbrace{\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}_*) \rangle}_{\text{Negative}} + o(t_k).$$

For t_k sufficiently small, we therefore have:

$$f(\boldsymbol{z}_k) < f(\boldsymbol{x}_*),$$

which contradicts the fact that $\boldsymbol{x}_*$ is a minimizer:

$$f(\boldsymbol{x}_*) \leq f(\boldsymbol{z}), \qquad \text{for all } \boldsymbol{z} \text{ in a neighborhood of } \boldsymbol{x}_*.$$

Hence, the assumption (16.5) is false, there is no such $\boldsymbol{d}$, so we conclude:

$$\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}_*) \rangle \geq 0, \qquad \text{for all } \boldsymbol{d} \in T_\Omega(\boldsymbol{x}_*).$$

16.2.1 Caution

The converse is not true. We can have $\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}_*) \rangle \geq 0$ without $\boldsymbol{x}_*$ being a local minimizer. For instance, consider:

$$\min f(\boldsymbol{x}) = y, \qquad \text{subject to } y \geq -x^2.$$

See Figure 16.1. The solution to the OP is unbounded, with $y_* = -\infty$.

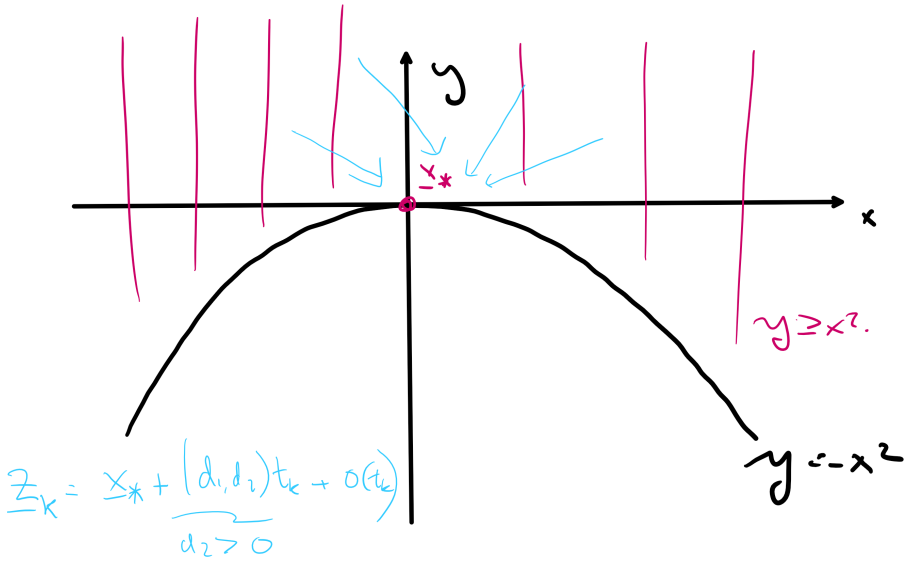


Figure 16.1: Counter-example showing $\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}_*) \rangle \geq 0$ but $\boldsymbol{x}_* = (0,0)^T$ is not a minimizer.

We compute the tangent cone at e.g. $\boldsymbol{x}_* = (0,0)^T$:

$$T_\Omega(\boldsymbol{x}_*) = \{(d_1, d_2) | d_2 \geq 0\}.$$

Also, $\nabla f(\boldsymbol{x}_*) = \mathbf{j}$, and for $\boldsymbol{d}$ in the tangent cone we have $\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}_*) \rangle = d_2 \geq 0$.

But $\boldsymbol{x}_* = (0,0)^T$ is not a minimum, it is not even a local minimum. For consider a point $\boldsymbol{x}_\alpha = (\alpha, -\alpha^2)^T$ on $\partial\Omega$. We have $f(\boldsymbol{x}_\alpha) = -\alpha^2$, thus:

$$f(\boldsymbol{x}_\alpha) < f(\boldsymbol{x}_*).$$

Also, $\boldsymbol{x}_\alpha$ is in the neighborhood of $\boldsymbol{x}_*$, as $\boldsymbol{x}_\alpha$ can be made 'close' to $\boldsymbol{x}_*$ by taking α sufficiently small. But $f(\boldsymbol{x}_\alpha) < f(\boldsymbol{x}_*)$, so $\boldsymbol{x}_*$ is not a local minimizer.

16.3 Farkas's Lemma and the Hyperplane Separation Theorem

We next look at **Farkas's Lemma**. We work with a modified veriosn of the lemma, this is to enable us to prove the KKT theorem. We will furthermore present a different proof of the lemma, different to what is in Nocedal and Wright. The motivation here is to explore some of the geometric reasoning behind constrained optimization. For these purposes, we introduce the Hyperplane Separation Theorem:

Theorem 16.2 (Hyperplane Separation Theorem) *Let A and B be two disjoint nonempty convex subsets of $\mathbb{R}^n$. Then there exists a plane which separates A and B. Mathematically, there exists a non-zero vector $\boldsymbol{n}$ and a constant c such that:*

$$\langle \boldsymbol{x}, \boldsymbol{n} \rangle \geq c, \qquad \langle \boldsymbol{y}, \boldsymbol{n} \rangle \leq c,$$

for all $\boldsymbol{x} \in A$ and $\boldsymbol{y} \in B$.

There is no proof in this module, but the idea is shown in Figure 16.2.

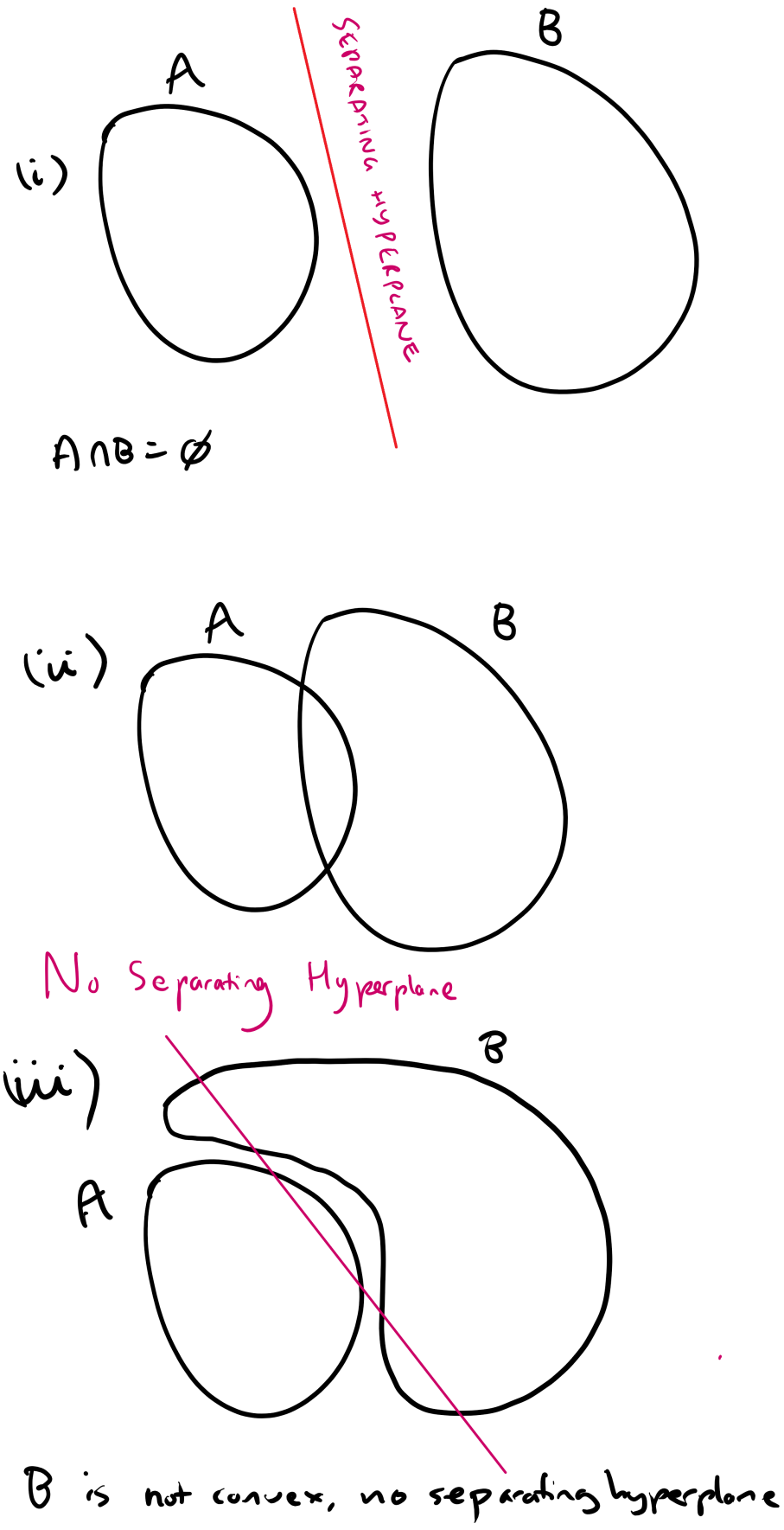


Figure 16.2: The idea behind the Hyperplane Separation Theorem

The Hyperplane Separation Theorem comes in many different flavours, the reader can check the reference

`http://aaa.princeton.edu/orf523`

for an enumeration. The following variation will be useful to us:

Theorem 16.3 (Hyperplane Separation Theorem, Special Case) *Let A and B be two disjoint nonempty convex subsets of $\mathbb{R}^n$, such that:*

- *A and B are closed;*
- *At least one of A and B is bounded.*

Then there exists a non-zero vector $\boldsymbol{n}$ and a constant c such that:

$$\langle \boldsymbol{x}, \boldsymbol{n} \rangle > c, \qquad \langle \boldsymbol{y}, \boldsymbol{n} \rangle < c,$$

for all $\boldsymbol{x} \in A$ and $\boldsymbol{y} \in B$.

We are now in a position to state and prove Farkas's Lemma.

Lemma 16.2 (Farkas) *Let*

$$\mathcal{K} = \{B\boldsymbol{y} | y_i \geq 0, \qquad i = 1, 2, \cdots, m\}$$

be a cone, where $B \in \mathbb{R}^{m \times n}$. Given any vector $\boldsymbol{g} \in \mathbb{R}^n$,

- *Either*

$$\boldsymbol{g} \in \mathcal{K} \tag{16.6a}$$

- *Or, there exists a vector $\boldsymbol{d} \in \mathbb{R}^n$ such that:*

$$\langle \boldsymbol{g}, \boldsymbol{d} \rangle < 0, \text{ and } [B^T \boldsymbol{d}]_i \geq 0, i = 1, 2, \cdots, m. \tag{16.6b}$$

For the idea behind Farkas's Lemma, see Figure 16.3.

Proof: We show first that Equation (16.6a) and (16.6b) can't hold simultaneously. Let $\boldsymbol{g} \in \mathcal{K}$. Hence,

$$\boldsymbol{g} = B\boldsymbol{y}, \qquad y_i \geq 0.$$

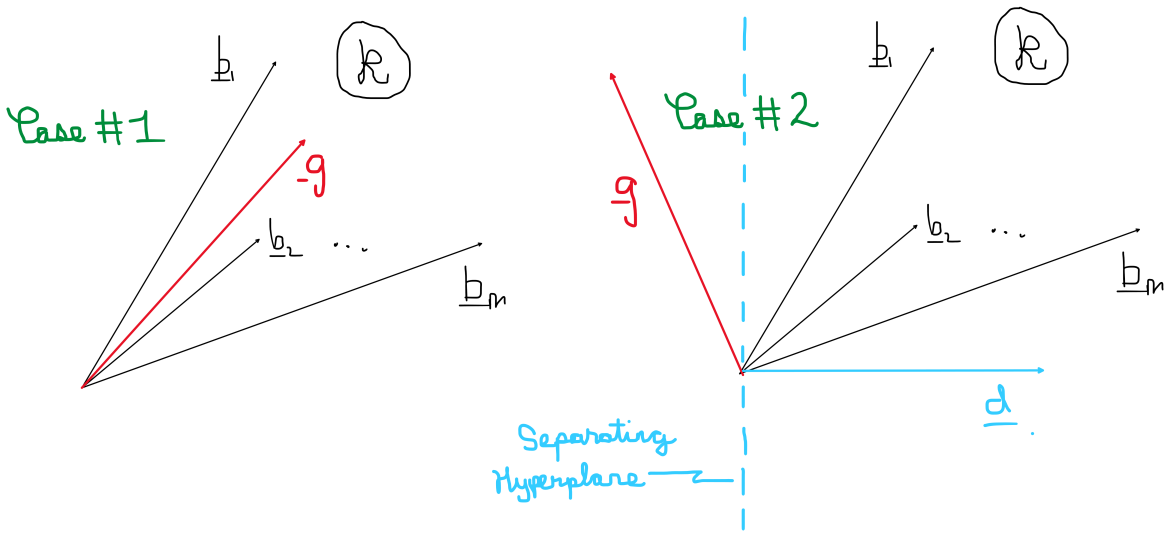


Figure 16.3: The idea behind Farkas's Lemma: either $\boldsymbol{g} \in \mathcal{K}$ (left) or there is a separating hyperplane (right)

If there exists a vector $\boldsymbol{d}$ with property (16.6b), then:

$$\begin{aligned} \langle \boldsymbol{g}, \boldsymbol{d} \rangle &< 0, \\ \langle \boldsymbol{g}, \boldsymbol{d} \rangle &= \langle B\boldsymbol{y}, \boldsymbol{d} \rangle, \\ &= \sum_{i=1}^{m} y_i [B^T \boldsymbol{d}]_i. \end{aligned}$$

But $[B^T \boldsymbol{d}]_i > 0$ and $y_i \geq 0$, giving $\langle \boldsymbol{g}, \boldsymbol{d} \rangle < 0$ and $\langle \boldsymbol{g}, \boldsymbol{d} \rangle \geq 0$ simultaneously, which is a contradiction. So Options #1 and #2 in Farkas's Lemma can't hold simultaneously.

We now show that at least one of the options always holds. If $\boldsymbol{g} \in \mathcal{K}$ we are done. So assume that $\boldsymbol{g} \notin \mathcal{K}$. Then we have two disjoint convex sets:

- First Set $= \{\boldsymbol{g}\}$, which is closed, bounded, and convex.
- Second Set $= \mathcal{K} = \{B\boldsymbol{y} \in \mathbb{R}^n | y_i \geq 0,\ i = 1, 2, \cdots, m\}$; as a cone, this set is is closed and convex[1]

By the Hyperplane Separation Theorem (Special Case), there exists a constant c and a vector $\boldsymbol{d} \in \mathbb{R}^n$ such that:

$$\langle \boldsymbol{d}, \boldsymbol{g} \rangle < c$$

and

$$\langle \boldsymbol{d}, \boldsymbol{s} \rangle > c \qquad \text{for all } \boldsymbol{s} \in \mathcal{K}.$$

[1] A closed set S is one that contains its own boundary. Specifically, if $\boldsymbol{x}_k$ is a sequence of points in S, and if $\boldsymbol{x}_k \to \boldsymbol{x}_*$, then $\boldsymbol{x}_* \in S$ also.

Now, the zero vector is in the cone $\mathcal{K}$, hence $\langle \boldsymbol{d}, 0\rangle > c$, hence $c < 0$. Hence,

$$\langle \boldsymbol{d}, \boldsymbol{g}\rangle < 0.$$

Furthermore, $\langle \boldsymbol{d}, \boldsymbol{s}\rangle > c$ for all $\boldsymbol{s} \in \mathcal{K}$, hence $\langle \boldsymbol{d}, B\boldsymbol{y}\rangle > c$, hence

$$\sum_{i=1}^{m} y_i [B^T \boldsymbol{d}]_i > c.$$

This is true for all $y_i \geq 0$. So replace y_i with $y_i\lambda$, where λ is positive:

$$\sum_{i=1}^{m} \lambda y_i [B^T \boldsymbol{d}]_i > c,$$

or

$$\sum_{i=1}^{m} y_i [B^T \boldsymbol{d}]_i > c/\lambda.$$

Now take $\lambda \to \infty$ to get:

$$\sum_{i=1}^{m} y_i [B^T \boldsymbol{d}]_i > 0.$$

This is true for all $y_i \geq 0$, hence

$$[B^T \boldsymbol{d}]_i \geq 0. \quad \blacksquare$$

16.3.1 Extension

Farkas's Lemma applies equally to a cone of type:

$$\mathcal{K} = \left\{ B\boldsymbol{y} + C\boldsymbol{w} \;\middle|\; \begin{array}{cc} y_i \in \mathbb{R}^+ \cup \{0\}, & i = 1, \cdots, m \\ w_i \in \mathbb{R}, & i = 1, \cdots p \end{array} \right\}, \tag{16.7}$$

where B and C are matrices of appropriate dimension. In that case, for any $\boldsymbol{g} \in \mathbb{R}^n$,

- Either

$$\boldsymbol{g} \in \mathcal{K} \tag{16.8a}$$

- Or, there exists a vector $\boldsymbol{d} \in \mathbb{R}^n$ such that:

$$\langle \boldsymbol{g}, \boldsymbol{d}\rangle < 0, \text{ and } [B^T \boldsymbol{d}]_i \geq 0, \text{ for } i = 1, 2, \cdots, m, \\ \text{and } [C^T \boldsymbol{d}]_i = 0, \text{ for } i = 1, 2, \cdots, p. \tag{16.8b}$$

16.3.2 Application

We now apply Farkas's lemma to the cone:

$$\begin{aligned} N &= \{ \sum_{i\in\mathcal{A}(\boldsymbol{x}_*)\cap\mathcal{I}} \lambda_i \nabla c_i(\boldsymbol{x}_*) + \sum_{i\in\mathcal{A}(\boldsymbol{x}_*)\cap\mathcal{E}} \lambda_i \nabla c_i(\boldsymbol{x}_*) | \lambda_i \geq 0 \text{ for } i \in \mathcal{A}(\boldsymbol{x}_*)\cap\mathcal{I}\}, \\ &= \{ \sum_{i\in\mathcal{A}(\boldsymbol{x}_*)} \lambda_i \nabla c_i(\boldsymbol{x}_*) | \lambda_i \geq 0 \text{ for } i \in \mathcal{A}(\boldsymbol{x}_*)\cap\mathcal{I}\}, \\ &= \{A^T\boldsymbol{\lambda} | \lambda_i \geq 0 \text{ for } \in \mathcal{A}(\boldsymbol{x}_*)\cap\mathcal{I}\}. \end{aligned}$$

This is a cone of type (16.7) Set $\boldsymbol{g} = \nabla f(\boldsymbol{x}_*)$. Then,

- Either $\nabla f(\boldsymbol{x}_*) \in N$ ($\implies \nabla f(\boldsymbol{x}_*) = A^T\boldsymbol{\lambda}$),
- Or there is a direction $\boldsymbol{d} \in \mathbb{R}^n$ such that

$$\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}_*)\rangle < 0,$$

and

$$[A\boldsymbol{d}]_i \geq 0, \text{ for } i \in \mathcal{A}(\boldsymbol{x}_*)\cap\mathcal{I} \text{ and } [A\boldsymbol{d}]_i = 0, \text{ for } i \in \mathcal{E}. \tag{16.9}$$

Condition (16.9) means that $\boldsymbol{d} \in \mathcal{F}_\Omega(\boldsymbol{x}_*)$.

16.4 Proof of KKT

We now prove the first-order necessary conditions for $\boldsymbol{x}_*$ to be a minimizer, the KKT conditions, or Theorem 16.1.

Let $\boldsymbol{x}_*$ be a local minimizer. We first of all show that there exist multipliers λ_i, $i \in \mathcal{A}(\boldsymbol{x}_*)$, such that:

$$\nabla f(\boldsymbol{x}_*) = \sum_{i\in\mathcal{A}(\boldsymbol{x}_*)} \lambda_i \nabla c_i(\boldsymbol{x}_*).$$

Lemma 16.1 tells us that

$$\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}_*)\rangle \geq 0, \qquad \text{for all } \boldsymbol{d} \in T_\Omega(\boldsymbol{x}_*).$$

But the LICQ holds, hence

$$\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}_*)\rangle \geq 0, \qquad \text{for all } \boldsymbol{d} \in \mathcal{F}_\Omega(\boldsymbol{x}_*).$$

Hence, Option #1 holds in Farkas's Lemma, hence there exist $\lambda_i \geq 0$ (for $i \in \mathcal{A}(\boldsymbol{x}_*)$) such that:

$$\nabla f(\boldsymbol{x}_*) = \sum_{i \in \mathcal{A}(\boldsymbol{x}_*)} \lambda_i \nabla c_i(\boldsymbol{x}_*).$$

We now define:

$$\lambda_i^* = \begin{cases} \lambda_i, & i \in \mathcal{A}(\boldsymbol{x}_*), \\ 0, & i \in \mathcal{I} \setminus \mathcal{A}(\boldsymbol{x}_*). \end{cases}$$

Hence,

- KKT1 follows immediately:

$$\nabla f(\boldsymbol{x}_*) = \sum_{i \in \mathcal{A}(\boldsymbol{x}_*)} \lambda_i \nabla c_i(\boldsymbol{x}_*).$$

- Since $\boldsymbol{x}_*$ is feasible by assumption, KKT2–3 are satisfied.
- We have:

$$\lambda_i^* \begin{cases} \lambda_i, & i \in \mathcal{A}(\boldsymbol{x} - *), \\ 0, & i \in \mathcal{I} \setminus \mathcal{A}(\boldsymbol{x}_*). \end{cases}$$

  Here, the λ_i's are positive or zero, since Option #1 holds in Farkas's lemma. Hence, $\lambda_i^* \geq 0$, for all $i \in \mathcal{I}$, and KKT 4 is shown.
- We have:

$$\begin{cases} i \in \mathcal{A}(\boldsymbol{x}_*) \cap \mathcal{E}: & c_i(\boldsymbol{x}_*) = 0 \\ i \in \mathcal{A}(\boldsymbol{x}_*) \cap \mathcal{I}: & c_i(\boldsymbol{x}_*) = 0, \\ i \in \mathcal{I} \setminus \mathcal{A}(\boldsymbol{x}_*): & \lambda_i^* = 0 \end{cases}$$

  hence $\lambda_i^* c_i(\boldsymbol{x}_*) = 0$, for all $i \in \mathcal{E} \cup \mathcal{I}$, and KKT5 is satisfied. ∎

Chapter 17

Constrained Optimization: Other Aspects

Overview

We look at some final key facts about constrained optimization:

- Second-order necessary and sufficient conditions.
- The dual problem.

We again work with the canonical constrained OP:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \text{ subject to } \begin{cases} c_i(\boldsymbol{x}) = 0, & i \in \mathcal{E}, \\ c_i(\boldsymbol{x}) \geq 0, & i \in \mathcal{I}. \end{cases} \tag{17.1}$$

and we again restate the KKT conditions valid when $\boldsymbol{x}_*$ is a solution of the OP:

Suppose that $\boldsymbol{x}_*$ is a minimizer of the OP (16.1)*. Furthermore, suppose that f and the c_i's are continuously differentiable, and that the LICQ holds at $\boldsymbol{x}_*$. Then, there exists a vector $\boldsymbol{\lambda}_*$ with components λ_i^*, with $i \in \mathcal{E} \cup \mathcal{I}$, such that the following conditions hold:*

$$\begin{aligned} \nabla_x \mathcal{L}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*) &= 0, && && \text{(17.2a)} \\ c_i(\boldsymbol{x}_*) &= 0, && i \in \mathcal{E}, && \text{(17.2b)} \\ c_i(\boldsymbol{x}_*) &\geq 0, && i \in \mathcal{I}, && \text{(17.2c)} \\ \lambda_i &\geq 0, && i \in \mathcal{I}, && \text{(17.2d)} \\ \lambda_i^* c_i(\boldsymbol{x}_*) &= 0, && i \in \mathcal{I} \cup \mathcal{E}. && \text{(17.2e)} \end{aligned}$$

As before, we refer to these conditions as KKT1–6.

17.1 Second-Order Conditions

We address first (without proof) the second-order conditions for $\boldsymbol{x}_*$ to **minimize** the OP (17.1). This is the constrained optimization equivalent of a second-derivative test. We first of all introduce the **critical cone**:

$$\mathcal{C}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*) = \left\{ \boldsymbol{w} \in \mathcal{F}_\Omega(\boldsymbol{x}_*) \;\middle|\; \begin{array}{ll} \boldsymbol{w} \cdot \nabla c_i(\boldsymbol{x}_*) = 0, & i \in \mathcal{A}(\boldsymbol{x}_*) \cap \mathcal{E} \\ \boldsymbol{w} \cdot \nabla c_i(\boldsymbol{x}_*) = 0, & i \in \mathcal{A}(\boldsymbol{x}_*) \cap \mathcal{I} \text{ and } \lambda_i^* > 0, \\ \boldsymbol{w} \cdot \nabla c_i(\boldsymbol{x}_*) \geq 0, & i \in \mathcal{A}(\boldsymbol{x}_*) \cap \mathcal{I} \text{ and } \lambda_i^* = 0 \end{array} \right\}. \tag{17.3}$$

Since $\lambda_i^* = 0$ for $i \in \mathcal{I} \backslash \mathcal{A}(\boldsymbol{x}_*)$, we have:

$$\lambda_i^* \boldsymbol{w} \cdot \nabla c_i(\boldsymbol{x}_*) = 0 \text{ for all } \boldsymbol{w} \in \mathcal{C}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*). \tag{17.4}$$

Suppose now that the KKT conditions are satisfied. We have:

$$\nabla f(\boldsymbol{x}_*) = \sum_{i \in \mathcal{A}(\boldsymbol{x}_*)} \lambda_i^* \nabla c_i(\boldsymbol{x}_*).$$

Dot both sides with $\boldsymbol{w} \in \mathcal{C}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*)$:

$$\boldsymbol{w} \cdot \nabla f(\boldsymbol{x}_*) = \sum_{i \in \mathcal{A}(\boldsymbol{x}_*)} \lambda_i^* \boldsymbol{w} \cdot \nabla c_i(\boldsymbol{x}_*) \overset{\text{Eq. (17.4)}}{=} 0.$$

This gives another view of the first-order necessary conditions for $\boldsymbol{x}_*$ to be a minimizer – the critical cone $\mathcal{C}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*)$ gives those directions in which we require the directional derivatives of f to vanish.

The critical cone also gives insights into second-order necessary and sufficient conditions for $\boldsymbol{x}_*$ to be a minimizer of the OP (16.1), which we state here finally (without proof).

Theorem 17.1 *Suppose that $\boldsymbol{x}_*$ is a local solution of the OP* (16.1) *and that the LICQs are satisfied. Let $\boldsymbol{\lambda}_*$ be the Lagrange multiplier vector for which the KKT conditions are satisfied. Then:*

$$\langle \boldsymbol{w}, \nabla^2_{xx} \mathcal{L}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*) \boldsymbol{w} \rangle \geq 0, \text{ for all } \boldsymbol{w} \in \mathcal{C}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*).$$

Here, $\nabla^2_{xx} \mathcal{L}$ denotes the Hessian of $\mathcal{L}$.

Conversely, we have a **sufficient condition** for $\boldsymbol{x}_*$ to be a local minimizer:

Theorem 17.2 *Suppose that for some feasible point $\boldsymbol{x}_*$ there is a Lagrange Multplier vector $\boldsymbol{\lambda}_*$ such that the KKT conditions are satisfied. Suppose also that:*

$$\langle \boldsymbol{w}, \nabla^2_{xx} \mathcal{L}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*) \boldsymbol{w} \rangle > 0, \text{ for all } \boldsymbol{w} \in \mathcal{C}(\boldsymbol{x}_*, \boldsymbol{\lambda}_*), \qquad \boldsymbol{w} \neq 0.$$

Then $\boldsymbol{x}_*$ is a strict local minimizer of the OP (17.1).

17.2 Duality

In this section we intrdouce an alternative posing of the OP (17.1). This is valid under some limitations:

- Inequality constraints only, with

$$\boldsymbol{c}(\boldsymbol{x}) = (c_1(\boldsymbol{x}), \cdots, c_m(\boldsymbol{x}))^T;$$

  and feasibility is expressed by $c_i(\boldsymbol{x}) \geq 0$, for each $i \in \{1, 2, \cdots, m\}$.
- f and $-c_i$ are convex functions on $\mathbb{R}^n$.

Thus, the special case of the OP (17.1) can be written as:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \text{ subject to } c_i(\boldsymbol{x}) \geq 0, \qquad i \in \{1, 2, \cdots, m\} \tag{17.5}$$

The idea is to introduce a general, unconstrained Lagrangian:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \langle \boldsymbol{\lambda}, \boldsymbol{c}(\boldsymbol{x}) \rangle.$$

We identify the dual objective function q:

$$\begin{aligned} q : \mathbb{R}^m &\to \mathbb{R}, \\ \boldsymbol{\lambda} &\mapsto q(\boldsymbol{\lambda}), \end{aligned}$$

where $q(\boldsymbol{\lambda})$ is defined as:

$$q(\boldsymbol{\lambda}) = \inf_{\boldsymbol{x}\in\mathbb{R}^n} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \tag{17.6}$$

For this to exist, we restrict the domain of $\boldsymbol{\lambda}$ to be:

$$\mathcal{D} = \{\boldsymbol{\lambda} \in \mathbb{R}^m | q(\boldsymbol{\lambda}) > -\infty\}.$$

Furthermore, the infimum in Equation (17.6) is unique: since $\mathcal{L}(\cdot, \boldsymbol{\lambda})$ is a convex function in its first variable ($= \boldsymbol{x}$), any local minimizer is guaranteed to be the global minimizer. The claim now is that finding the minimizer of the OP (17.5) is equivalent to solving the **dual problem**

$$\max_{\boldsymbol{\lambda}\in\mathbb{R}^m} q(\boldsymbol{\lambda}), \text{ subject to } \boldsymbol{\lambda} \geq 0. \tag{17.7}$$

17.2.1 Worked Example

We solve the OP

$$\min \tfrac{1}{2}\left(x^2+y^2\right) \text{ subject to } x-1\geq 0.$$

Simply by inspection, we see that $\boldsymbol{x}_* = (1,0)^T$ is the minimizer.

However, we can instead construct the Lagrangian:

$$\mathcal{L}(\boldsymbol{x},\lambda) = \tfrac{1}{2}(x^2+y^2) - \lambda(x-1), \qquad \boldsymbol{x} = (x,y)^T.$$

This is a linear combination of linear and quadratic functions in $\boldsymbol{x}$; as such, $\mathcal{L}$ is convex in $\boldsymbol{x}$.

We minimize $\mathcal{L}$ over $\boldsymbol{x}$ to obtain $q(\lambda)$. The minimizer is obtained by setting $\nabla_x \mathcal{L} = 0$. This gives:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x} &= 0 \implies x-\lambda = 0, \\ \frac{\partial \mathcal{L}}{\partial y} &= 0 \implies y = 0. \end{aligned}$$

Hence, $x=\lambda$ and $y=0$. Thus,

$$q(\lambda) = \mathcal{L}(x=\lambda, y=0, \lambda) = -\tfrac{1}{2}\lambda^2 + \lambda.$$

The dual problem is thus:

$$\max_{\lambda\in\mathbb{R}} q(\lambda), \qquad \lambda = 0.$$

We compute $\mathrm{d}q/\mathrm{d}\lambda = -\lambda + 1$. Thus, $\mathrm{d}q/\mathrm{d}\lambda = 0 \implies \lambda = 1$. A quick plot of $q(\lambda)$ shows that $\lambda = 1$ is a maximum. Thus, the solution of the dual problem is $\lambda = 1$. Going back to $x=\lambda$ and $y=0$, this gives the solution of the OP as $x=1$ and $y=0$, hence $\boldsymbol{x}_* = (1,0)^T$, as expected.

17.3 Preliminary Results

For the dual problem (17.7) to have a solution in the general case, we require:

- $q(\boldsymbol{\lambda})$ to be a concave function[1];
- $\mathcal{D}$ to be a convex domain.

We prove that both these properties apply. Hence, let $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ be any two points in $\mathbb{R}^m$, and form the line segment

$$\boldsymbol{\lambda}(t) = (1-t)\boldsymbol{\lambda} + t\boldsymbol{\mu}, \qquad t\in[0,1].$$

[1] Yes, concave. Not a typo.

We have:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}(t)) = (1-t)\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) + t\mathcal{L}(\boldsymbol{x}, \boldsymbol{\mu}).$$

We take the infimum of both sides, and we use the fact that:

$$\inf(A+B) \geq \inf(A) + \inf(B).$$

Hence:

$$\begin{aligned} \inf_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}(t)) &= \inf_{\boldsymbol{x}} \left[(1-t)\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) + t\mathcal{L}(\boldsymbol{x}, \boldsymbol{\mu})\right], \\ &\geq (1-t)\inf_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) + t \inf_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\mu}). \end{aligned}$$

Hence,

$$q(\boldsymbol{\lambda}(t)) \geq (1-t)q(\boldsymbol{\lambda}) + t\, q(\boldsymbol{\mu}), \qquad t \in [0,1].$$

Furthermore, if $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are both in $\mathcal{D}$, then:

$$q(\boldsymbol{\lambda}(t)) \geq (1-t)q(\boldsymbol{\lambda}) + t\, q(\boldsymbol{\mu}) > -\infty, \qquad t \in [0,1].$$

Hence, $q(\boldsymbol{\lambda}(t)) \in \mathcal{D}$ also, for all $t \in [0,1]$ and hence, $\mathcal{D}$ is a convex domain.

We also have a final preliminary result which will help us in the next section:

Theorem 17.3 *For any feasible $\overline{\boldsymbol{x}}$ and any $\overline{\boldsymbol{\lambda}} \geq 0$, we have:*

$$q(\overline{\boldsymbol{\lambda}}) \leq f(\overline{\boldsymbol{x}}).$$

Proof:

$$\begin{aligned} q(\overline{\boldsymbol{\lambda}}) &= \inf_{\boldsymbol{x}} \left[f(\boldsymbol{x}) - \langle \overline{\boldsymbol{\lambda}}, \boldsymbol{c}(\boldsymbol{x})\rangle\right], \\ &\leq f(\overline{\boldsymbol{x}}) - \langle \overline{\boldsymbol{\lambda}}, \boldsymbol{c}(\boldsymbol{x})\rangle. \end{aligned}$$

But $\overline{\boldsymbol{x}}$ is feasible, hence $c_i(\overline{\boldsymbol{x}}) \geq 0$. Also, $\overline{\boldsymbol{\lambda}} \geq 0$. Hence:

$$\begin{aligned} q(\overline{\boldsymbol{\lambda}}) &\leq f(\overline{\boldsymbol{x}}) - \langle \overline{\boldsymbol{\lambda}}, \boldsymbol{c}(\boldsymbol{x})\rangle, \\ &\leq f(\overline{\boldsymbol{x}}). \quad \blacksquare \end{aligned}$$

17.4 Key Theorems

We know that a local minimizer $\boldsymbol{x}_*$ which solves the OP satisfies the KKT conditions. We now show that such a minimizer also satisfies the dual problem. Thus, instead of solving the KKT conditions and hence constructing a candidate local minimizer, we can solve the dual problem instead. This is sometimes easier. Establishing this equivalence requires two theorems, which we prove here.

Theorem 17.4 *Suppose that $\overline{\boldsymbol{x}}$ is a solution of the OP. Then , any $\overline{\boldsymbol{\lambda}}$ for which $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}})$ satisfies the KKT conditions is a solution of the dual problem.*

Proof: Suppose that $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}})$ satisfies the KKT conditions. Since $\mathcal{L}(\cdot, \overline{\boldsymbol{\lambda}})$ is convex, we have, by the second part of Theorem 2.8,

$$\mathcal{L}(\boldsymbol{x}, \overline{\boldsymbol{\lambda}}) \geq \mathcal{L}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}}) + \langle \underbrace{\nabla_x \mathcal{L}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}})}_{=0}, (\boldsymbol{x} - \overline{\boldsymbol{x}}) \rangle.$$

The term with the underbrace is zero, by the KKT conditions. Hence:

$$\mathcal{L}(\boldsymbol{x}, \overline{\boldsymbol{\lambda}}) \geq \mathcal{L}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}}), \qquad \forall \boldsymbol{x}. \tag{17.8}$$

We have:

$$\begin{aligned} q(\overline{\boldsymbol{\lambda}}) &= \inf_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \overline{\boldsymbol{\lambda}}), \\ &\overset{\text{Eq. (17.8)}}{=} \mathcal{L}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}}), \\ &= f(\overline{\boldsymbol{x}}) - \langle \overline{\boldsymbol{\lambda}}, \boldsymbol{c}(\overline{\boldsymbol{x}}) \rangle, \\ &\overset{\text{KKT}}{=} f(\overline{\boldsymbol{x}}). \end{aligned}$$

Hence,

$$q(\overline{\boldsymbol{\lambda}}) = f(\overline{\boldsymbol{x}}).$$

But, from Theorem 17.3,

$$q(\overline{\boldsymbol{\lambda}}) \leq f(\overline{\boldsymbol{x}}), \text{ for all } \overline{\boldsymbol{x}} \text{ feasible.}$$

Thus, $\overline{\boldsymbol{\lambda}}$ solves the dual problem. ∎

We have shown here that if $\overline{\boldsymbol{x}}$ solves the OP, then if a $\overline{\boldsymbol{\lambda}}$ can be found such that $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}})$ solves the KKT conditions, then $\overline{\boldsymbol{\lambda}}$ is a solution of the dual problem. That is:

$$\left\{ \begin{array}{c} \overline{\boldsymbol{x}} \text{ solves the OP and} \\ (\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}}) \text{ solve the KKT conditions} \end{array} \right\} \implies \overline{\boldsymbol{\lambda}} \text{ solves the dual problem.}$$

We would like to be able to go the other way around:

$$\left\{ \begin{array}{c} \overline{\boldsymbol{x}} \text{ solves the OP and} \\ \overline{\boldsymbol{\lambda}} \text{ solves the dual problem} \end{array} \right\} \implies (\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}}) \text{ satisfy the KKT conditions.}$$

We show this now, under some extra assumptions:

Theorem 17.5 *Suppose that the following conditions hold:*

- f *and* $-c_i$ *are convex and continuously differentiable;*
- $\overline{\boldsymbol{x}}$ *solves the OP and the LICQ hold at* $\overline{\boldsymbol{x}}$*;*
- $\widehat{\boldsymbol{\lambda}}$ *solves the dual problem;*
- $\inf_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \widehat{\boldsymbol{\lambda}})$ *is attained at* $\widehat{\boldsymbol{x}}$*;*
- $\mathcal{L}(\cdot, \widehat{\boldsymbol{\lambda}})$ *is strictly convex.*

Then:

$$\widehat{\boldsymbol{x}} = \overline{\boldsymbol{x}} \text{ and } f(\overline{\boldsymbol{x}}) = \mathcal{L}(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{\lambda}}).$$

Proof: Assume for contradiction that $\widehat{\boldsymbol{x}} \neq \overline{\boldsymbol{x}}$. Since $\overline{\boldsymbol{x}}$ is a minimizer, it satisfies the KKT conditions. So there exists a $\overline{\boldsymbol{\lambda}} \geq 0$ such that:

$$\mathcal{L}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}}) \overset{\text{Thm. (17.4)}}{=} q(\overline{\boldsymbol{\lambda}}) = q(\widehat{\boldsymbol{\lambda}}) = \mathcal{L}(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{\lambda}}). \tag{17.9}$$

Here, we have used the following fact:

$$\widehat{\boldsymbol{x}} = \text{arg min}_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \widehat{\boldsymbol{\lambda}}),$$

from the theorem statement. Hence:

$$\nabla_x \mathcal{L}(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{\lambda}}) = 0.$$

By strict convexity of $\mathcal{L}(\cdot, \widehat{\boldsymbol{\lambda}})$, we have:

$$\mathcal{L}(\overline{\boldsymbol{x}}, \widehat{\boldsymbol{\lambda}}) > \mathcal{L}(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{\lambda}}) + \langle \underbrace{\nabla_x \mathcal{L}(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{\lambda}})}_{=0}, (\overline{\boldsymbol{x}} - \widehat{\boldsymbol{x}}) \rangle.$$

Hence,

$$\mathcal{L}(\overline{\boldsymbol{x}}, \widehat{\boldsymbol{\lambda}}) > \mathcal{L}(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{\lambda}})$$

But $\mathcal{L}(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{\lambda}}) = \mathcal{L}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}})$, by Equation (17.9). Hence,

$$\mathcal{L}(\overline{\boldsymbol{x}}, \widehat{\boldsymbol{\lambda}}) > \mathcal{L}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}}).$$

Since $f(\overline{\boldsymbol{x}}) = f(\overline{\boldsymbol{x}})$, this gives:

$$-\langle \widehat{\boldsymbol{\lambda}}, \boldsymbol{c}(\overline{\boldsymbol{x}})\rangle > -\langle \overline{\boldsymbol{\lambda}}, \boldsymbol{c}(\overline{\boldsymbol{x}})\rangle.$$

The RHS is zero, by the complementarity condition (KKT5). Thus,

$$\langle \widehat{\boldsymbol{\lambda}}, \boldsymbol{c}(\overline{\boldsymbol{x}})\rangle < 0.$$

Since $\widehat{\boldsymbol{\lambda}} \geq 0$ and $\boldsymbol{c}(\overline{\boldsymbol{x}}) \geq 0$, this is a contradiction. Hence, $\widehat{x} = \overline{\boldsymbol{x}}$, as claimed. ∎

Chapter 18

Global Optimization via Simulated Annealing

Overview

So far – when looking at solutions of the unconstrained OP, $\boldsymbol{x}_* = \arg\min f(\boldsymbol{x})$, we have been concerned with iterative methods that stop when a local solution is found. Such methods do not distinguish between a local minimum and the global minimum. Therefore, in this Chapter, we look at one particular method that can find a global minimum. This is called **Simulated Annealing**, and is inspired by ideas from Physics.

18.1 Physics

Imagine a system with n continuous degrees of freedom. The 'phase space' of the system is $\mathbb{R}^n$. Suppose that the system's energy is $E(\boldsymbol{x})$, this is a function that maps states in the system (vectors in $\mathbb{R}^n$) to real numbers. The probability that the state of the system is to be found in a small region of phase space of volume $\mathrm{d}^n x$, centred at $\boldsymbol{x}$ is:

$$\mathrm{d}\mathbb{P} = p(\boldsymbol{x})\mathrm{d}^n\boldsymbol{x},$$

hence, $p(\boldsymbol{x})$ is a probability distribution function, with unit normalization:

$$\int_{\mathbb{R}^n} p(\boldsymbol{x})\mathrm{d}^n\boldsymbol{x} = 1.$$

The energy of the system is therefore:

$$\overline{E} = \int_{\mathbb{R}^n} E(\boldsymbol{x}) p(\boldsymbol{x}) \mathrm{d}nx.$$

The entropy of the system is given by the Botlzmann formula:

$$S = -\int_{\mathbb{R}^n} p(\boldsymbol{x}) \log p(\boldsymbol{x}) \mathrm{d}^n x.$$

18.1.1 The Boltzmann Distribution

W seek to maximize the entropy while maintaining the average value of the energy at a constant value, $\overline{E} = \int E(\boldsymbol{x}) p(\boldsymbol{x}) \mathrm{d}^n x$. Hence, we maximize the function

$$\widetilde{S} = -\int p(\boldsymbol{x}) \log p(\boldsymbol{x}) \mathrm{d}^n x - \beta \left(\int E(\boldsymbol{x}) p(\boldsymbol{x}) \mathrm{d}^n x - \overline{E} \right) + \alpha \left(\int_{\mathbb{R}^n} p(\boldsymbol{x}) \mathrm{d}^n \boldsymbol{x} - 1 \right)$$

(we omit the subscript on the integral from now on, as the region of integration is clear). We compute $\delta \widetilde{S}$:

$$\delta \widetilde{S} = -\int \left[\log p(\boldsymbol{x}) + 1 \right] \delta p - \beta \int E(\boldsymbol{x}) \delta p \, \mathrm{d}^n x + \alpha \int \delta p \, \mathrm{d}^n x.$$

We require:

$$\frac{\delta \widetilde{S}}{\delta p} = 0.$$

Hence:

$$\log p = -\beta E + \alpha - 1,$$

or $p(\boldsymbol{x}) = \mathrm{e}^{\alpha - 1} \mathrm{e}^{-\beta E(\boldsymbol{x})}$. The Lagrange multiplier α can be eliminated by imposing that $\int p(\boldsymbol{x}) \mathrm{d}^n x = 1$, hence:

$$p(\boldsymbol{x}) = \frac{\mathrm{e}^{-\beta E(\boldsymbol{x})}}{\int \mathrm{e}^{-\beta E(\boldsymbol{x})} \mathrm{d}^n x}$$

We identify:

$$Z = \int \mathrm{e}^{-\beta E(\boldsymbol{x})} \mathrm{d}^n x.$$

Hence,

$$p(\boldsymbol{x}) = \frac{\mathrm{e}^{-\beta E(\boldsymbol{x})}}{Z} = p_T(E(\boldsymbol{x})). \tag{18.1}$$

Equation (18.1) describes the **Boltzmann Distribution**. The parameter β is the Lagrange multiplier that enforces the constant average energy. We further identify $T = 1/\beta$ as the temperature, hence:

$$p(\boldsymbol{x}) = \frac{\mathrm{e}^{-E(\boldsymbol{x})/T}}{Z} = p_T(E(\boldsymbol{x})), \tag{18.2}$$

where $p_T(E) = \mathrm{e}^{-\beta E}/Z$.

Furthermore, we have:

$$\begin{aligned} S_{max} &= -\int p(\boldsymbol{x}) \log p(\boldsymbol{x}) \mathrm{d}x + 0, \\ &= -\int \frac{\mathrm{e}^{-\beta E(\boldsymbol{x})}}{Z} \left[-\beta E(\boldsymbol{x}) - \log Z\right], \\ &= \beta \overline{E} + \log Z. \end{aligned}$$

Also,

$$\overline{E} = -\frac{\partial \log Z}{\partial \beta},$$

and

$$\begin{aligned} \frac{\partial S_{max}}{\partial \overline{E}} &= \beta + \overline{E}\frac{\partial \beta}{\partial \overline{E}} + \frac{\partial}{\partial \overline{E}} \log Z, \\ &= \beta + \overline{E}\frac{\partial \beta}{\partial \overline{E}} + \frac{\partial \log Z}{\partial \beta}\frac{\partial \beta}{\partial \overline{E}}, \\ &= \beta + \overline{E}\frac{\partial \beta}{\partial \overline{E}} + (-\overline{E})\frac{\partial \beta}{\partial \overline{E}}, \\ &= \beta, \end{aligned}$$

which gives the fundamental temperature-entropy relationship:

$$\frac{1}{T} = \frac{\partial S_{max}}{\partial \overline{E}}.$$

18.1.2 The Quench

As the system is cooled to zero absolute temperature (the 'quench'), the Boltzman distribution tends to a delta function centred at the minimum energy. Assuming a unique global minimum, there is therefore only one allowed state at zero temperature (the minimizer). Hence, the system entropy also tends to zero. Furthermore, the mean energy $\overline{E}$ tends to the minimum value:

$$\overline{E} \to E_{min}, \qquad \text{as } T \to 0.$$

The idea of simulated annealing therefore is to view optimization of a cost function $E(\boldsymbol{x})$ as equivalent to the process of quenching a physical system to absolute zero temperature. The physical energy tends to zero if and only if the cost function $E(\boldsymbol{x})$ attains the global minimum. Thus, the challenge for a simulated-annealing algorithm is to simulate the quench of the physical system to absolute zero.

18.2 Simulated Annealing – Algorithm

The idea in Simulated Annealing is to start with an initial 'temperature' T_0 and an initial guess for the state of the system $\boldsymbol{x}^{(0)}$. A proposal to move the system into a new $\boldsymbol{x}^{(1)}$ is generated, using random-number generation. This mimics the stochastic nature of real physical systems ('Brownian Motion'). If the proposed new state reduces the cost function, it is accepted with probability 1. Thus, let:

$$E^{(1)} = E(\boldsymbol{x}^{(1)}), \qquad E^{(0)} = E(\boldsymbol{x}^{(0)}), \qquad \Delta E = E^{(1)} - E^{(0)}.$$

If $\Delta E < 0$, the updated guess decreases energy (cost function), and is accepted. However, to stop the system from getting stuck in a local minimum, a proposal that increases the energy will from time to time be accepted. Hence, if $\Delta E > 0$, we accept the proposal (which increases the energy) with a probability given by the Boltzmann distribution:

$$\mathbb{P}\left(\text{Accept } \boldsymbol{x}^{(0)} \to \boldsymbol{x}^{(1)}\right) = \begin{cases} 1, & \text{if } \Delta E < 0, \\ \mathrm{e}^{-\Delta E/T_0}, & \text{if } \Delta E > 0. \end{cases}$$

We also denote this **transition function** by $h(\Delta E)$. We continue thus, for a certain number of iterations, whereupon the temperature is lowered to T_1, and the algorithm continues again. The method for lowering the temperature after a set number of iterations is called the **annealing schedule**.

18.2.1 Detailed balance

The SA schedule satisfies detailed balance, in the sense that:

$$p_T(E^{(k)})\mathbb{P}\left(\boldsymbol{x}^{(k)} \to \boldsymbol{x}^{(k+1)}\right) = p_T(E^{(k+1)})\mathbb{P}\left(\boldsymbol{x}^{(k+1)} \to \boldsymbol{x}^{(k)}\right).$$

Without the change of temperature, detailed balance is sufficient to establish that all states of the system can be sampled. With the change of temperature, all states of the system can still be sampled, however, this must be done carefully and gradually.

18.3 Statement of Algorithm

Algorithm 7 Simulated Annealing Algorithm

Choose an initial guess $\boldsymbol{x}^{(0)}$. Initialize $\boldsymbol{x} = \boldsymbol{x}^{(0)}$.

Select the temperature change counter $k = 0$

Select a cooling schedule T_k

Select an initial temperature $T = T_0 \geq 0$

Select a repetition schedule M_k, that defines the number of iterations executed at each temperature T_k

while Stopping criterion is not met **do**

Set repetition counter $m = 0$

while $m < M_k$ **do**

Generate a new state $\boldsymbol{x}'$

Calculate $\Delta E = E(\boldsymbol{x}') - E(\boldsymbol{x})$.

If $\Delta E \leq 0$, accept the new state, $\boldsymbol{x} \leftarrow \boldsymbol{x}'$ with probability 1.

If $\Delta E > 0$, accept the new state, $\boldsymbol{x} \leftarrow \boldsymbol{x}'$, with probability $\mathrm{e}^{-\Delta E/T_k}$.

$m \leftarrow m + 1$

end while

end while

The algorithm results in $M_0 + M_1 + \cdots + M_k$ total iterations being executed, where k corresponds to the value for T_k at which some stopping criterion is met – for example, a pre-specified total number of iterations has been exceeded, or a solution of a certain quality has been found. In addition, if $M_k = 1$ for all k, then the temperature changes at each iteration

A sample (very simple) Matlab code to implement the algorithm is shown in the following code listings.

```
function [best_x, best_E, x_vec, E_vec]=mySA0()

% Function for computing the global minimum of:
%
% y=sin(x)/(x^2+10);

% Initialize x:
x=5;

maxit_outer = 250;   % Maximum Number of Iterations
maxit_inner = 15;    % Maximum Number of Sub-iterations

T0 = 1;          % Initial Temperature
alpha = 0.95;        % Temperature Reduction Rate

% Initialize temperature:
T=T0;

```

```
% Initial energy:
E=my_cost_fn(x);

best_E=E;
best_x=x;

ctr=1;
x_vec=0*(1:maxit_outer*maxit_inner);
E_vec=0*(1:maxit_outer*maxit_inner);

for k_outer=1:maxit_outer

    for k_inner = 1:maxit_inner

        % Generate new guess using a normal distribution:
        x_new=normrnd(x, 2*T, 1);
        E_new=my_cost_fn(x_new);

        delta_E=E_new-E;

        % If delta_E<=0, then accept the new guess with probability 1.
        if(delta_E<=E)
            x=x_new;
            E=E_new;
        else
            % If delta_E>0, then accept the new guess with a probability
            % exp(-delta_E/T).

            prob = exp(-delta_E/T);
            if(rand <= prob)
                x = x_new;
                E = E_new;
            end
        end

        if(E<best_E)
            best_E=E;
            best_x=x;
        end

        x_vec(ctr)=x;
        E_vec(ctr)=E;
        ctr=ctr+1;

    end

    % Update the temperature:
    T = alpha*T;
    % T=5/(log(k_outer)+1);

    display(strcat('outer iteration=',num2str(k_outer),'; best value=',num2str(best_E),'; best x=',num2str(best_x)));

end

end

% ************************************************************************

function y=my_cost_fn(x)
    % y=(x-1)^2
    y=sin(x)/(x^2+10);
end
```

The updated guess is selected to be a random perturbation away from the current guess, with the perturbation chosen from a normal distribution, the standard deviation of which is $2T$, where T is the current temperature. This is then related to the annealing schedule $T_k = T_0/(\ln(k) + 1)$, more details of which are discussed in the final section below. Other annealing schedules are possible, for instance, $T_{k+1} = \alpha T_k$, where $0 < \alpha < 1$, the main idea here is that the temperature is gradually

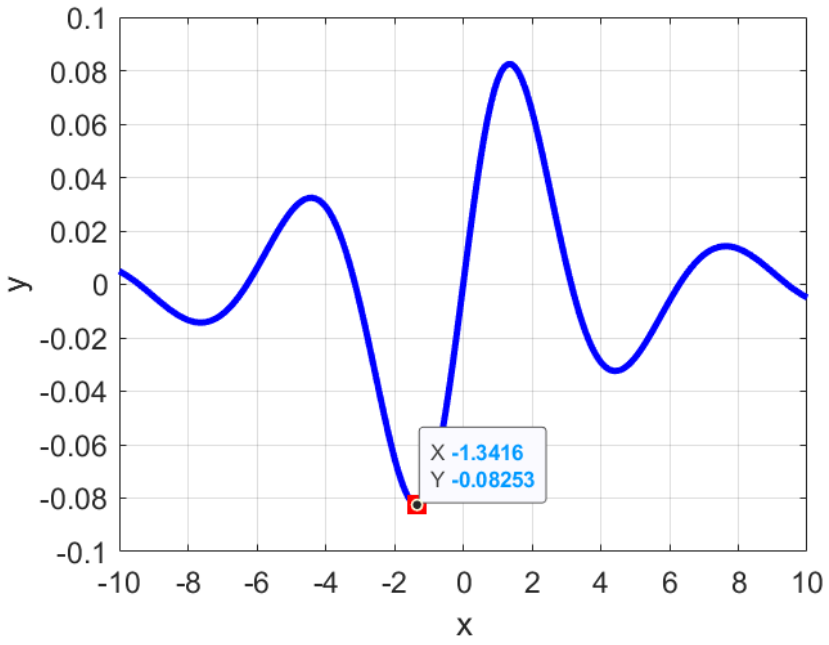


Figure 18.1: The cost function with multiple local minima; the SA algorithm successfully picks out the global minimum

lowered as the algorithm converges to a global minimum. Finally, results of running the algorithm are shown in Figure 18.1: the algorithm successfully picks out the global minimum.

18.4 Annealing Schedule (Boltzmann)

In this section, we assume that new proposals are generated using random-nummber generation: if $\boldsymbol{x}_1$ is the new proposal and $\boldsymbol{x}_0$ is the old proposal, then:

$$\boldsymbol{x}_1 \sim N(\boldsymbol{x}_0, \sigma_k),$$

where, for the present purposes, σ_k^2 is taken to be T_k, hence

$$\sigma_k = \sqrt{T_k}.$$

This approach is referred to as **Boltzmann annealing**, it is different to the numerical algorithms used in previous sections where σ_k was taken to be $2T_k$ (fast annealing).

The probability that the new proposal is in a region $\mathcal{R}$ of phase space is:

$$\mathbb{P}(\text{New proposal in } \mathcal{R}) = \int_{\mathcal{R}} p(\boldsymbol{x}) \mathrm{d}^n x,$$

where

$$p(\boldsymbol{x}) = \frac{1}{(2\pi\sigma_k^2)^{n/2}} \mathrm{e}^{-\|\boldsymbol{x}-\boldsymbol{x}_0\|_2^2/2\sigma_k^2}.$$

We propose an annealing schedule

$$T_k \leq \frac{T_0}{\ln k}, \tag{18.3}$$

where T_0 is a parameter that is 'sufficiently large'. Thus, the aim of this section is to show that this is a 'good' annealing schedule, for Boltzmann annealing.

For this purpose, we look at the probability g_k that the new proposal is to be found in a small volume ΔV around the global minimum, we denote this region by $\mathcal{R}(\boldsymbol{x}_*, \Delta V)$:

$$g_k = \mathbb{P}(\text{New proposal in } \mathcal{R}(\boldsymbol{x}_*, \Delta V)).$$

Hence,

$$g_k = \int_{\mathcal{R}(\boldsymbol{x}_*, \Delta V)} p(\boldsymbol{x}) \mathrm{d}^n x.$$

We have:

$$\begin{aligned}
g_k &= \mathbb{P}(\text{New proposal in } \mathcal{R}(\boldsymbol{x}_*, \Delta V)), \\
&= \frac{1}{(2\pi\sigma_k^2)^{n/2}} \int_{\mathcal{R}(\boldsymbol{x}_*, \Delta V))} \mathrm{e}^{-\|\boldsymbol{x}-\boldsymbol{x}_0\|_2^2/2\sigma_k^2} \, \mathrm{d}^n x, \\
&= \left[\frac{1}{(2\pi\sigma_k^2)^{1/2}} \int_{x_{*1}-(\Delta x/2)}^{x_{*1}+\Delta x/2} \mathrm{e}^{-(x_1-x_{01})^2/2\sigma_k^2} \, \mathrm{d}x_1 \right] \times \cdots \\
&\qquad \times \left[\frac{1}{(2\pi\sigma_k^2)^{1/2}} \int_{x_{*n}-(\Delta x/2)}^{x_{*n}+\Delta x/2} \mathrm{e}^{-(x_n-x_{0n})^2/2\sigma_k^2} \, \mathrm{d}x_n \right], \\
&\approx \left[\frac{1}{(2\pi\sigma_k^2)^{1/2}} \mathrm{e}^{-(x_{*1}-x_{01})^2/2\sigma_k^2} \Delta x \right] \times \cdots \times \left[\frac{1}{(2\pi\sigma_k^2)^{1/2}} \mathrm{e}^{-(x_{*n}-x_{0n})^2/2\sigma_k^2} \Delta x \right], \\
&= \frac{(\Delta x)^n}{(2\pi\sigma_k^2)^{n/2}} \mathrm{e}^{-\|\boldsymbol{x}_*-\boldsymbol{x}_0\|_2^2/2\sigma_k^2}, \\
&= \frac{\Delta V}{(2\pi\sigma_k^2)^{n/2}} \mathrm{e}^{-\|\boldsymbol{x}_*-\boldsymbol{x}_0\|_2^2/2\sigma_k^2},
\end{aligned}$$

where we have used the Trapezoidal Rule,

$$\int_{a-\Delta x/2}^{a+\Delta x/2} f(x) \mathrm{d}x \approx f(a)\Delta x,$$

valid for smooth functions.

As it turns out, it is better to look at the probability that the new proposal will **not** be within ΔV of the global minimum, this will be $1 - g_k$. Thus, the probability that the new proposals will **never**

be within ΔV of the global minimum is:

$$\prod_k (1 - g_k).$$

It now suffices to show that the probability $\prod_k (1 - g_k)$ tends to zero, as the number of temperature updates tends to infinity; this will establish that the probability of generating the global minimum tends to one:

$$\text{To Show: } \prod_k (1 - g_k) \to 0, \qquad \text{as } k \to \infty.$$

We can take logarithms on both sides, then we have to show:

$$\text{To Show: } \sum_k \log(1 - g_k) \to -\infty, \qquad \text{as } k \to \infty.$$

As g_k is pre-multiplied by an arbitrary factor ΔV, we take ΔV sufficeintly small, such that $\log(1 - g_k) \approx -g_k$, by Taylor approxmimation. Thus, it suffices to show:

$$\text{To Show: } \sum_k g_k \to \infty, \qquad \text{as } k \to \infty. \tag{18.4}$$

Hence we need to show:

$$\begin{aligned}
\sum_k g_k &\to \infty, && \text{as } k \to \infty, \\
\text{or } \sum_k \frac{1}{(2\pi T_k)^{n/2}} \mathrm{e}^{-\|\boldsymbol{x}_* - \boldsymbol{x}_0\|_2^2/(2T_k)} &\to \infty, && \text{as } k \to \infty, \\
\text{or } \sum_k \left[(\log k)^{n/2}\right] \mathrm{e}^{-(\log k)\|\boldsymbol{x}_* - \boldsymbol{x}_0\|_2^2/(2T_0)} &\to \infty, && \text{as } k \to \infty,
\end{aligned}$$

We choose T_0 'sufficiently large', specifically:

$$\frac{\|\boldsymbol{x}_* - \boldsymbol{x}_0\|_2^2}{2T_0} \leq 1.$$

In the 'tail' of the series, with $k \geq k_0$, we have:

$$\begin{aligned}
\sum_{k=k_0}^{\infty} \left[(\log k)^{n/2}\right] \mathrm{e}^{-(\log k)\|\boldsymbol{x}_* - \boldsymbol{x}_0\|_2^2/(2T_0)} &\geq \sum_{k=k_0}^{\infty} \left[(\log k)^{n/2}\right] \mathrm{e}^{-(\log k)}, \\
&\geq \sum_{k=k_0}^{\infty} \left[(\log k)^{n/2}\right] \frac{1}{k}, \\
&\geq \sum_{k=k_0}^{\infty} \frac{1}{k} = \infty,
\end{aligned}$$

where the result here is a divergent series because the harmonic series $\sum_{k=1}^{\infty}(1/k)$ is the divergent harmonic series. Thus, the sum in Equation (18.4) diverges, as required. This establishes the result that the annealing schedule (18.3) will converge to the global minimum, albeit that infinite iterations may be required.

Chapter 19

Introduction to Artificial Neural Networks

Overview

We introduce the idea of Artificial Neural Networks as a particularly efficient way of approximating a function, given some training data. We outline how the function can be fitted to the data – referred to as training, and we show how the fitting amounts to an optimization problem involving gradient descent. The computation of the gradients is made very efficient by way of so-called *back-propagation*.

19.1 Context

Consider for example an input vector $\boldsymbol{x} \in \mathbb{R}^2$ (components x_1 and x_2). We have in mind here a classification problem, where each $\boldsymbol{x} \in \mathbb{R}^2$ is associated with some Type A or Type B. Hence, $\boldsymbol{y}(\boldsymbol{x}) = (1,0)^T$ for type A and $\boldsymbol{y}(\boldsymbol{x}) = (0,1)^T$ for type B. It is then desired to find find the 'decision boundary', which is a curve in space separating regions of type A from regions of type B (see Figure 19.1).

In applications, $\boldsymbol{x}$ could be latitude and longitude, and

$$\boldsymbol{y}(\boldsymbol{x}) = \begin{cases} (1,0)^T \text{ if } \boldsymbol{x} \text{ is mineral-rich,} \\ (0,1)^T \text{ if } \boldsymbol{x} \text{ is mineral-poor.} \end{cases} \tag{19.1}$$

A curve enclosing mineral-rich regions then tells us where mineral exploration should occur. In precise terms, this would be the contour where $y_1(\boldsymbol{x}) = y_2(\boldsymbol{x})$. Deriving such a curve will require data – which would be obtained by taking samples from various places and determining if the samples are mineral rich or poor. This is the *training data*. Say samples are taken at points $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N$ locations

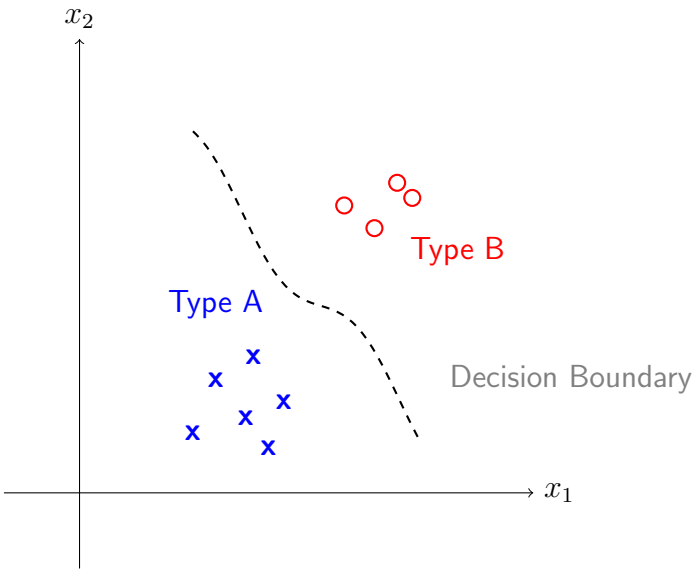


Figure 19.1: Classification problem, with training data.

yielding scores $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_N$. Recalling the formulation in Section 1, we fit a functional form

$$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{p}),$$

to the training data, where f is a function $f : \mathbb{R}^2 \to \mathbb{R}^2$, and $\boldsymbol{p}$ are parameters. We then estimate the parameters by minimizing the loss function

$$\widetilde{\mathcal{L}}(\boldsymbol{p}) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{\|f(\boldsymbol{x}_i, \boldsymbol{p}) - \boldsymbol{y}_1\|_2^2}_{=C_i(\boldsymbol{p})}. \tag{19.2}$$

In this way, we reduce the problem of Machine Learning to solving $\nabla_p \widetilde{\mathcal{L}} = 0$. This is a standard unconstrained optimization problem, so it is relatively straightforward to solve. For instance, using steepest descent, we have the following iterative scheme:

$$\boldsymbol{p}^{N+1} = \boldsymbol{p}^N - \eta \nabla_p \widetilde{\mathcal{L}}(\boldsymbol{p}^N), \tag{19.3}$$

where η is the stepsize – called the **learning rate** in ML.

Leaving aside the poor convergence rate of Steepest Descent, a problem with line search is the computation of the derivatives $\nabla_p \widetilde{\mathcal{L}}$. It's not a problem of there is a simple analytical formula for $\widetilde{\mathcal{L}}$, and if the number of parameters is small. But if the number of observations n is in the billions and the number of parameters is in the millions (as in modern data-science problems), then this is a huge issue. So the point of departure for Machine Learning, away from 'traditional' regression models and non-linear least squares, is the speeding-up of the computation of $\nabla_p \widetilde{\mathcal{L}}$. This is done in two complementary ways:

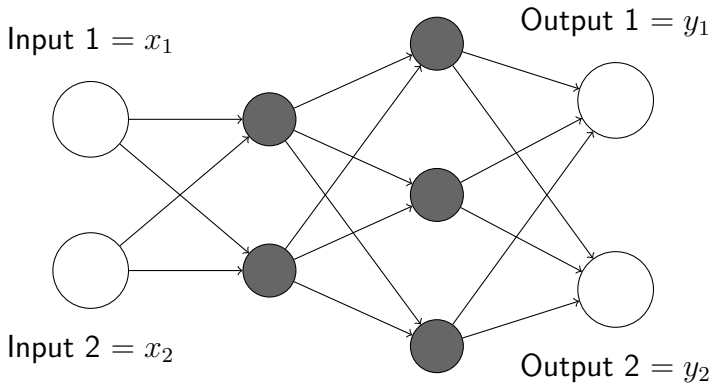


Figure 19.2: Sample ANN with two hidden layers

1. A particular choice of $f(\boldsymbol{x};\boldsymbol{p})$, which enables the extremely efficient computation of the derivatives $\nabla_p\widetilde{\mathcal{L}}$;
2. Stochastic Gradient Descent.

We look at the first point here, and postpone discussion of Stochastic Gradient Descent to Chapter 20.

19.2 Artificial Neural Networks

Artificial Neural Networks (ANNS) enable the extremely efficient computation of derivatives of $\nabla_p\widetilde{\mathcal{L}}$, and hence, represent an extremely efficient form of numerical optimization. The idea is to approximate $f(\boldsymbol{x};\boldsymbol{p})$ by a composition of:

- Activation functions;
- Linear functions – so-called weights and biases.

This idea was originally proposed by Frank Rosenblatt [7] as a model of how the human brain works – as a network of nodes which are either on or off ('neurons'), and connections to nodes ('synapses'). A small neural network for the classification problem (19.1) is shown in Figure 19.2.

The idea is to model $\boldsymbol{y}(\boldsymbol{x})$ as a composition:

i. The input layer (first layer) $\boldsymbol{a}^{(1)} \equiv \boldsymbol{x}$ ...
ii. Gets transformed according to:

$$\boldsymbol{z}^{(2)} = W^{(2)}\boldsymbol{a}^{(1)} + \boldsymbol{b}^{(2)}...$$

iii. Which is then moved to the second layer and activated according to:

$$\boldsymbol{a}^{(2)} = \sigma(\boldsymbol{z}^{(2)})...$$

iv. Which is then transformed according to:

$$\boldsymbol{z}^{(3)} = W^{(3)}\boldsymbol{a}^{(2)} + \boldsymbol{b}^{(3)}...$$

v. Which is then moved to third layer and activated according to:

$$\boldsymbol{a}^{(3)} = \sigma(\boldsymbol{z}^{(3)})\,...$$

vi. Which is then transformed according to:

$$\boldsymbol{z}^{(4)} = W^{(4)}\boldsymbol{a}^{(4)} + \boldsymbol{b}^{(4)}...$$

vii. Which is then moved to the fourth layer and activated according to:

$$\boldsymbol{a}^{(4)} = \sigma(\boldsymbol{z}^{(4)}).$$

viii. As the fourth layer is the output layer, we have $\boldsymbol{y} = \boldsymbol{a}^{(4)}$.

A few observations are in order.

- The matrices $W^{(2)}, W^{(3)}, W^{(4)}$, vectors $\boldsymbol{z}^{(2)}, \boldsymbol{z}^{(3)}, \boldsymbol{z}^{(4)}$ etc. all have a well-defined dimension. In particular:
  - $\boldsymbol{z}^{(n)}$:
  $$\boldsymbol{z}^{(2)} \in \mathbb{R}^2, \qquad \boldsymbol{z}^{(3)} \in \mathbb{R}^3, \qquad \boldsymbol{z}^{(4)} \in \mathbb{R}^2.$$
  - $W^{(n)}$:
  $$W^{(2)} \in \mathbb{R}^{2\times 3}, \qquad W^{(3)} \in \mathbb{R}^{3\times 2}, \qquad W^{(4)} \in \mathbb{R}^{2\times 2}.$$
  - $\boldsymbol{b}^{(n)}$:
  $$\boldsymbol{b}^{(2)} \in \mathbb{R}^2, \qquad \boldsymbol{b}^{(3)} \in \mathbb{R}^3, \qquad \boldsymbol{b}^{(4)} \in \mathbb{R}^2.$$
  - $\boldsymbol{a}^{(n)}$:
  $$\boldsymbol{a}^{(2)} \in \mathbb{R}^2, \qquad \boldsymbol{a}^{(3)} \in \mathbb{R}^3, \qquad \boldsymbol{a}^{(4)} \in \mathbb{R}^2.$$
- $\sigma(\cdot)$ is defined element-wise, such that
  $$\sigma\left((\alpha_1, \cdots, \alpha_n)^T\right) = \begin{pmatrix} \sigma(\alpha_1) \\ \vdots \\ \sigma(\alpha_n) \end{pmatrix}$$
  for all $(\alpha_1, \cdots, \alpha_n)^T \in \mathbb{R}^n$.

Putting all this together, we have:

$$\begin{aligned}
\boldsymbol{y}(\boldsymbol{x}) &\approx \sigma(\boldsymbol{z}^{(4)}),\\
&= \sigma\left(W^{(4)}\boldsymbol{a}^{(3)}+\boldsymbol{b}^{(4)}\right),\\
&= \sigma\left[W^{(4)}\sigma\left(W^{(3)}\boldsymbol{a}^{(2)}+\boldsymbol{b}^{(3)}\right)+\boldsymbol{b}^{(4)}\right],\\
&= \sigma\Big\{W^{(4)}\sigma\left[W^{(3)}\sigma\left(W^{(2)}\boldsymbol{a}^{(1)}+\boldsymbol{b}^{(2)}\right)+\boldsymbol{b}^{(3)}\right]+\boldsymbol{b}^{(4)}\Big\},\\
&= \sigma\Big\{W^{(4)}\sigma\left[W^{(3)}\sigma\left(W^{(2)}\boldsymbol{x}+\boldsymbol{b}^{(2)}\right)+\boldsymbol{b}^{(3)}\right]+\boldsymbol{b}^{(4)}\Big\}.
\end{aligned}$$

Thus, the function f which models the relationship $\boldsymbol{y}=f(\boldsymbol{x},\boldsymbol{p})$ is obtained as a composition of linear transformations and activations functions, specifically

$$f(\boldsymbol{x},\boldsymbol{p})=\sigma\Big\{W^{(4)}\sigma\left[W^{(3)}\sigma\left(W^{(2)}\boldsymbol{x}+\boldsymbol{b}^{(2)}\right)+\boldsymbol{b}^{(3)}\right]+\boldsymbol{b}^{(4)}\Big\},$$

where the parameters are

$$\boldsymbol{p}=\Big\{W^{(2)},W^{(3)},W^{(4)},\boldsymbol{b}^{(2)},\boldsymbol{b}^{(3)},\boldsymbol{b}^{(4)}\Big\}.$$

Going back up to the list of dimensions of the various matrices and vectors, we wee that there are 23 different parameters. This means there are 23 different (highly complex) partial derivatives $\nabla_p\widetilde{\mathcal{L}}$ to compute, which is an immense computational task. To simplify this task, we look at back-propagation.

19.3 Back-Propagation

The idea of back-propagation is that the derivatives $\nabla_p\widetilde{\mathcal{L}}$ can be computed iteratively for an ANN, using the Chain Rule. To see this, we will go over to explicitly computing

$$\frac{\partial C}{\partial b_j^{(\ell)}},\qquad \frac{\partial C}{\partial W_{jk}^{(\ell)}},$$

where ℓ is labelling the layer, with $\ell\in\{2,\cdots,L\}$, and we suppress the subscript i on the cost-function component $C_i(\boldsymbol{p})$. We let:

$$\delta_j^{(\ell)}=\frac{\partial C}{\partial z_j^{(\ell)}}.$$

We also introduce the Hadamard pointwise multiplication of vectors:

$$(\boldsymbol{x}\odot\boldsymbol{y})_i=x_iy_i,\qquad \text{for all } \boldsymbol{x},\boldsymbol{y}\in\mathbb{R}^n.$$

We have the following theorem:

Theorem 19.1 *The following expressions for the partial derivatives are true:*

$$\boldsymbol{\delta}^{(L)} = \sigma'(\boldsymbol{z}^{(L)}) \odot \left(\boldsymbol{a}^{(L)} - \boldsymbol{y}\right), \tag{19.4}$$

$$\boldsymbol{\delta}^{(\ell)} = \sigma'(\boldsymbol{z}^{(\ell)}) \odot \left[(W^{(\ell+1)})^T \boldsymbol{\delta}^{(\ell+1)}\right], \tag{19.5}$$

for $\ell \in \{2, \cdots, L-1\}$*, and:*

$$\frac{\partial C}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)}, \tag{19.6}$$

$$\frac{\partial C}{\partial W_{jk}^{(\ell)}} = \delta_j^{(\ell)} a_k^{(\ell-1)}, \tag{19.7}$$

for $\ell \in \{2, \cdots, L\}$.

Proof: We prove each of these statements in turn. We start with:

$$\begin{aligned} \delta_j^{(L)} &= \frac{\partial C}{\partial z_j^{(L)}}, \\ &= \frac{\partial}{\partial z_j^{(L)}} \left\{ \tfrac{1}{2} \| f(\boldsymbol{x}) - \boldsymbol{y} \|_2^2 \right\}. \end{aligned}$$

Here, properly, we should write $f(\boldsymbol{x}; \boldsymbol{p})$ for the approximating function and $\boldsymbol{y}_i$ for the i^{th} observation; however, we suppress these indices for brevity. Hence, we have:

$$\begin{aligned} \delta_j^{(L)} &= \frac{\partial}{\partial z_j^{(L)}} \sum_{\alpha=1}^{2} \left(\tfrac{1}{2} f_\alpha(\boldsymbol{x}) f_\alpha(\boldsymbol{x}) - f_\alpha(\boldsymbol{x}) y_\alpha + \cdots \right), \\ &= \sum_{\alpha=1}^{2} \left[f_\alpha(\boldsymbol{x}) - y_\alpha \right] \frac{\partial f_\alpha(\boldsymbol{x})}{\partial z_j^{(L)}}, \\ &= \sum_{\alpha=1}^{2} \left[f_\alpha(\boldsymbol{x}) - y_\alpha \right] \frac{\partial a_\alpha(\boldsymbol{x})}{\partial z_j^{(L)}}. \end{aligned}$$

Continuing thus:

$$\begin{aligned} \delta_j^{(L)} &= \sum_{\alpha=1}^{2} \left[f_\alpha(\boldsymbol{x}) - y_\alpha \right] \frac{\partial}{\partial z_j^{(L)}} \sigma(z_\alpha^{(L)}), \\ &= \sum_{\alpha=1}^{2} \left[f_\alpha(\boldsymbol{x}) - y_\alpha \right] \frac{\partial}{\partial z_j^{(L)}} \sigma'(z_\alpha) \delta_{\alpha,j}, \\ &= \left[f_j(\boldsymbol{x}) - y_j \right] \sigma'(z_j^{(L)}), \\ &= \left(a_j - y_j \right) \sigma'(z_j^{(L)}). \end{aligned}$$

Hence, Equation (19.4) is shown. ∎

Next, we have:

$$\begin{aligned}
\delta_j^{(\ell)} &= \frac{\partial C}{\partial z_j^{(\ell)}}, \\
&\overset{\text{Chain Rule}}{=} \sum_{\alpha=1}^{n_{\ell+1}} \frac{\partial C}{\partial z_\alpha^{\ell+1}} \frac{\partial z_\alpha^{(\ell+1)}}{\partial z_j^{(\ell)}}, \\
&= \sum_{\alpha=1}^{n_{\ell+1}} \frac{\partial C}{\partial z_\alpha^{(\ell+1)}} \frac{\partial}{\partial z_j^{(\ell)}} \left[\sum_{\beta=1}^{n_\ell} W_{\alpha\beta}^{(\ell+1)} \sigma(z_\beta^{(\ell)}) + b_\beta^{(\ell+1)} \right], \\
&= \sum_{\alpha=1}^{n_{\ell+1}} \delta_\alpha^{(\ell+1)} \sum_{\beta=1}^{n_\ell} W_{\alpha\beta}^{(\ell+1)} \sigma'(z_\beta^{(\ell)}) \frac{\partial z_\beta^{(\ell)}}{\partial z_j^{(\ell)}}, \\
&= \sum_{\alpha=1}^{n_{\ell+1}} \delta_\alpha^{(\ell+1)} \sum_{\beta=1}^{n_\ell} W_{\alpha\beta}^{(\ell+1)} \sigma'(z_\beta^{(\ell)}) \delta_{\beta,j} \\
&= \sum_{\alpha=1}^{n_{\ell+1}} \delta_\alpha^{(\ell+1)} W_{\alpha j}^{(\ell+1)} \sigma'(z_j^{(\ell)}), \\
&= \sum_{\alpha=1}^{n_{\ell+1}} \sigma'(z_j^{(\ell)}) \left[\left(W^{(\ell+1)}\right)^T \boldsymbol{\delta}^{(\ell+1)} \right]_j.
\end{aligned}$$

Hence, in vector form:

$$\boldsymbol{\delta}^{(\ell)} = \sigma'(\boldsymbol{z}^{(\ell)}) \odot \left[\left(W^{(\ell+1)}\right)^T \boldsymbol{\delta}^{(\ell+1)} \right].$$

Hence, Equation (19.5) is shown. ∎

Next, we have:

$$\frac{\partial C}{\partial b_j^{(\ell)}} = \sum_{\alpha=1}^{n_\ell} \frac{\partial C}{\partial z_\alpha^{(\ell)}} \frac{\partial z_\alpha^{(\ell)}}{\partial b_j^{(\ell)}}.$$

But

$$z_\alpha^{(\ell)} = \left[W^{(\ell)} \sigma(\boldsymbol{z}^{(\ell-1)}) \right]_\alpha + b_\alpha^{(\ell)}.$$

Hence:

$$\frac{\partial z_\alpha^{(\ell)}}{\partial b_j^{(\ell)}} = \delta_{\alpha j},$$

and:

$$\begin{aligned}\frac{\partial C}{\partial b_j^{(\ell)}} &= \sum_{\alpha=1}^{n_\ell} \frac{\partial C}{\partial z_\alpha^{(\ell)}} \delta_{\alpha j}, \\ &= \frac{\partial C}{\partial z_j^{(\ell)}}. \\ &= \delta_j^{(\ell)},\end{aligned}$$

Hence, Equation (19.6) is shown. ∎

Finally, we have

$$\frac{\partial C}{\partial W_{jk}^{(\ell)}} = \sum_{\alpha=1}^{n_\ell} \frac{\partial C}{\partial z_\alpha^{(\ell)}} \frac{\partial z_\alpha^{(\ell)}}{\partial W_{jk}^{(\ell)}}.$$

But

$$\begin{aligned}z_\alpha^{(\ell)} &= \left[W^{(\ell)} \sigma(\boldsymbol{z}^{(\ell-1)})\right]_\alpha + b_\alpha^{(\ell)}, \\ &= \sum_{\beta=1}^{n_{\ell-1}} W_{\alpha\beta}^{(\ell)} \sigma(z_\beta^{(\ell-1)}) + b_\alpha^{(\ell)}.\end{aligned}$$

Hence:

$$\begin{aligned}\frac{\partial z_\alpha^{(\ell)}}{\partial W_{jk}^{(\ell)}} &= \sum_{\beta=1}^{n_{\ell-1}} \delta_{\alpha j} \delta_{\beta k} \sigma(z_\beta^{(\ell-1)}), \\ &= \delta_{\alpha j} \sigma(z_j^{(\ell-1)}), \\ &= \delta_{\alpha j} a_j^{(\ell-1)}.\end{aligned}$$

So, back to:

$$\begin{aligned}\frac{\partial C}{\partial W_{jk}^{(\ell)}} &= \sum_{\alpha=1}^{n_\ell} \frac{\partial C}{\partial z_\alpha^{(\ell)}} \frac{\partial z_\alpha^{(\ell)}}{\partial W_{jk}^{(\ell)}}, \\ &= \sum_{\alpha=1}^{n_\ell} \frac{\partial C}{\partial z_\alpha^{(\ell)}} \delta_{\alpha j} a_j^{(\ell-1)}, \\ &= \sum_{\alpha=1}^{n_\ell} \delta_\alpha^{(\ell)} \delta_{\alpha j} a_j^{(\ell-1)}, \\ &= \delta_j^{(\ell)} a_j^{(\ell-1)}.\end{aligned}$$

Hence, Equation (19.6) is shown. This completes the proof of Equations (19.4)–(19.7). ∎

Instead of the Hadamard product, we introduce diagonal matrices,

$$D^{(\ell)} = \begin{pmatrix} \sigma'(z_1^{(\ell)}) & 0 & \cdots & 0 \\ 0 & \sigma'(z_2^{(\ell)}) & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & \sigma'(z_{n_\ell}^{(\ell)}) \end{pmatrix} \in \mathbb{R}^{n_\ell \times n_\ell}.$$

Hence, Equations (19.4)–(19.5) can be re-written as:

$$\boldsymbol{\delta}^{(L)} = D^{(L)} \left(\boldsymbol{a}^{(L)} - \boldsymbol{y} \right), \tag{19.8a}$$

$$\boldsymbol{\delta}^{(\ell)} = D^{(\ell)} \left(W^{(\ell+1)} \right)^T \boldsymbol{\delta}^{(\ell+1)}, \qquad \ell \in \{2, \cdots L-1\}. \tag{19.8b}$$

Hence, we can telescope these products to get:

$$\boldsymbol{\delta}^{(\ell)} = D^{(\ell)} \left(W^{(\ell+1)} \right)^T \cdots D^{(L-1)} \left(W^{(L)} \right)^T D^{(L)} \left(\boldsymbol{a}^{(L)} - \boldsymbol{y} \right).$$

Also to note, from Equation (19.7) we see that $\partial C / \partial W_{jk}^{(\ell)}$ is a matrix which can be formed from an outer product:

$$\frac{\partial C}{\partial W^{(\ell)}} = \boldsymbol{\delta}^{(\ell)} \otimes \boldsymbol{a}^{(\ell-1)} \in \mathbb{R}^{n_\ell \times n_\ell},$$

with components

$$\frac{\partial C}{\partial W_{jk}^{(\ell)}} = \delta_j^{(\ell)} a_j^{(\ell-1)}.$$

19.3.1 Pseudocode

See Algorithm 19.3.1.

Algorithm 8 Backpropagation Algorithm

Input: Training data $\boldsymbol{x}^{\{k\}}$ and $y(\boldsymbol{x}^{\{k\}}$, with $k \in \{1, 2, \cdots, n\}$.
Output: Weights $W^{[l]}$ and biases $b^{[l]}$, with $l \in \{2, \cdots, L\}$.

for counter $= 1$ to N_{iter} **do**
    Choose k uniformly at random from $\{1, 2, \ldots, n\}$
    $\boldsymbol{x}^{\{k\}}$ is current training data point
    $\boldsymbol{a}^{[1]} = \boldsymbol{x}^{\{k\}}$

    Forward pass
    for $l = 2$ to L **do**
        $\boldsymbol{z}^{[l]} = W^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}$
        $\boldsymbol{a}^{[l]} = \sigma(\boldsymbol{z}^{[l]})$
        $D^{[l]} = \mathrm{diag}(\sigma'(\boldsymbol{z}^{[l]}))$
    end for

    Backward pass
    $\boldsymbol{\delta}^{[L]} = D^{[L]}(\boldsymbol{a}^{[L]} - y(\boldsymbol{x}^{\{k\}}))$
    for $l = L - 1$ downto 2 **do**
        $\boldsymbol{\delta}^{[l]} = D^{[l]}(W^{[l+1]})^T \delta^{[l+1]}$
    end for

    for $l = L$ downto 2 **do**
        $W^{[l]} = W^{[l]} - \eta\, \boldsymbol{\delta}^{[l]}(a^{[l-1]})^T$
        $b^{[l]} = b^{[l]} - \eta\, \boldsymbol{\delta}^{[l]}$
    end for
end for

19.3.2 Results

We take the training data in Figure 22.3 and attempt to construct a decision boundary separating type A and type B, using an ANN with the network architecture shown already in Figure 19.2. We

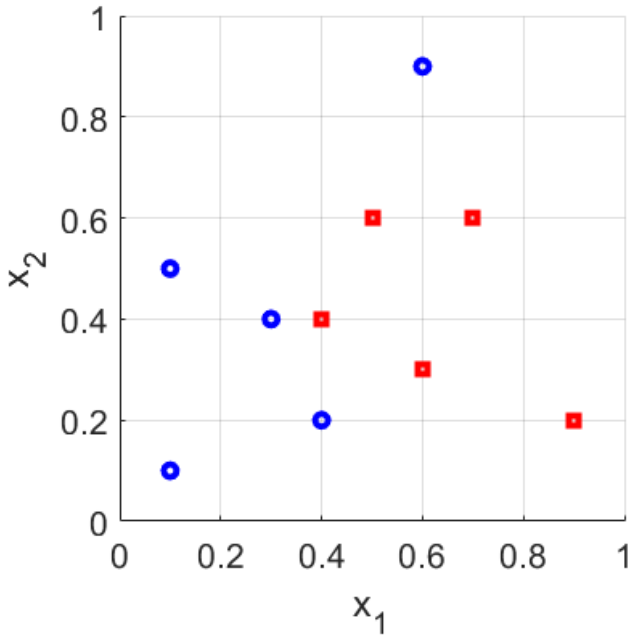


Figure 19.3: Training data for the ANN

use the Matlab code from Reference [6] (specifically, `netbp.m`). This is a basic instructional code, and the network architecture is 'hard coded', meaning that the forward and backward sweeps are very explicitly specified, without what would be a very obvious use of 'for' loops. However, the results of this exercise are impressive, and are shown in Figure 19.4 – Figure 19.5. Once the network has been trained and the optimal values of the weights and biases have been found, it is necessary to evalualte the approximate cost function $f(x_1, x_2) \in \mathbb{R}^2$ at each point in $[0, 1] \times [0, 1]$ and hence to construct the decision boundary $f_1(x_1, x_2) = f_2(x_1, x_2)$. We have developed our own code for these purposes, shown in the listings at the end of this section.

19.3.3 Comparison with built-in method

We compare the results in Figure 19.4 with the results of implementing Matlab's built-in functions for training artificial neural networks. The results are qualitatively the same but not quantitatively the same. The reasons for this are manifold:

- As the cost function is not convex in the vector of parameters, there is not necessarily a single global minimum. Matlab's optimization algorithm therefore has its own stopping criteria, which will come into effect when the cost function is close to a local (possibly non-unique

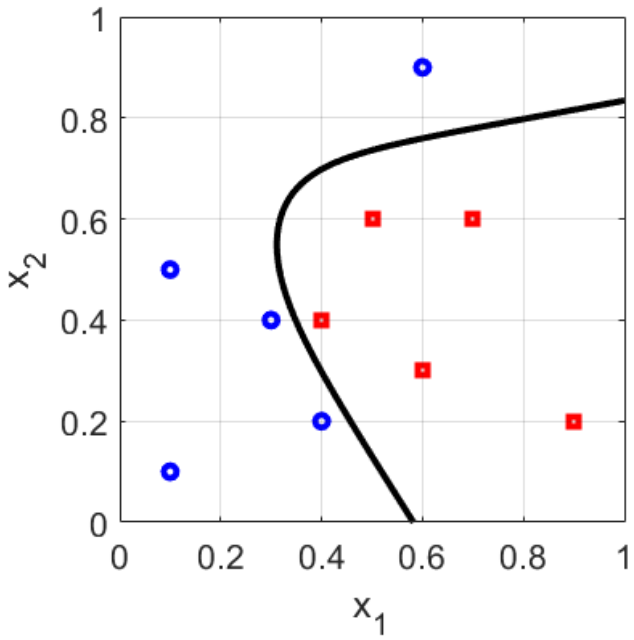


Figure 19.4: Decision boundary obtained using the ANN. Step length for the SGD algorithm: $\eta = 0.05$. Number of iterations: 10^6.

minimum), which may be different from the local minimum found using the hand-coded method.

- Indeed, Matlab's method (as with the hand-coded method) is stochastic, in the sense that the initial conditions are selected at random, and (if used) the stochastic gradient-descent algorithm samples different regions of parameter space each time it is used. Therefore again, Matlab's optimization algorithm may be converging to a local minimum different from the local minimum found using the hand-coded method.
- It might be best to compute an average decision boundary, averaged over many instances of the neural network.

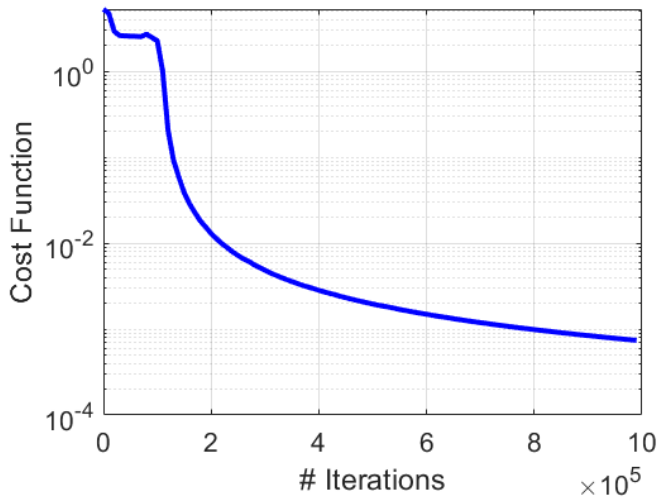


Figure 19.5: Decay of the cost function as a function of iteration number

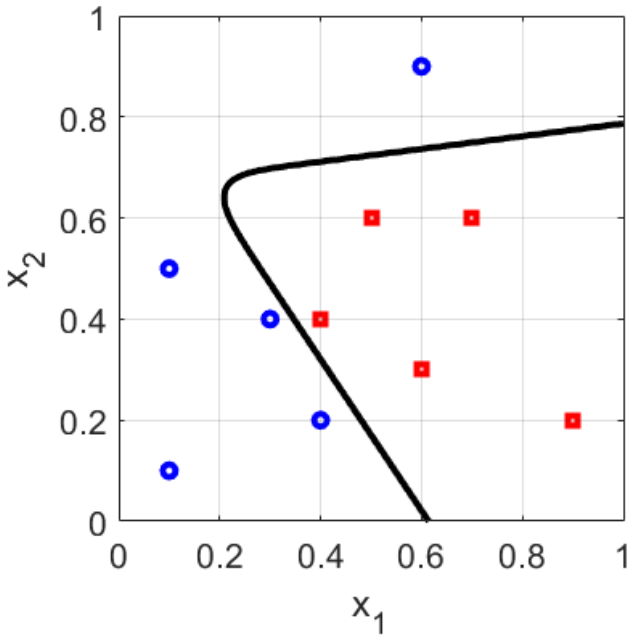


Figure 19.6: Decision boundary obtained using the ANN. ANN generated using Matlab's `feedforwardnet` function. Training with Matlab's `train` function. Options selected to maximize agreement with hand-coded ANN. Listings at the end of this section.

19.3.4 Listings

```
function [X,Y,Fx,Fy]=plot_decision_bdy()

temp=load('costvec.mat');

b2=temp.b2;
b3=temp.b3;
b4=temp.b4;

W2=temp.W2;
W3=temp.W3;
W4=temp.W4;

xx=0:0.01:1;
yy=0:0.01:1;

X=zeros(length(xx),length(yy));
Y=X;
Fx=X;
Fy=X;

for i=1:length(xx)
    for j=1:length(yy)
        x1_val=xx(i);
        x2_val=yy(j);

        x=[x1_val,x2_val]';

        % Forward pass
        a2 = activate(x,W2,b2);
        a3 = activate(a2,W3,b3);
        a4 = activate(a3,W4,b4);

        Fx_val=a4(1);
        Fy_val=a4(2);

        X(i,j)=x1_val;
        Y(i,j)=x2_val;

        Fx(i,j)=Fx_val;
        Fy(i,j)=Fy_val;
    end
end

contour(X,Y,Fx-Fy,[0 0],'LineWidth',3,'color','black')
hold on
% Overlay with training data:
[x1,x2,~]=saved_training_data();

% First five training data are type A - label as circles

for i=1:5
    z_point=x1(i)+sqrt(-1)*x2(i);
    plot(z_point,'o','Color','blue','LineWidth',3)
end
```

```

% First five training data are type B - label as squares

for i=6:10
    z_point=x1(i)+sqrt(-1)*x2(i);
    plot(z_point,'sq','Color','red','LineWidth',3)
end

xlabel('x_1')
ylabel('x_2')

set(gca,'fontsize',16)
grid on
axis equal
set(gca,'xtick',0:0.2:1)
set(gca,'ytick',0:0.2:1)
hold off

drawnow
```

```

[x1,x2,Yd]=saved_training_data();

n=length(x1);
Xd=zeros(2,n);
for i=1:n
    Xd(1,i)=x1(i);
    Xd(2,i)=x2(i);
end


net = feedforwardnet([2 3]);
% Default optimization method:
% Levenberg-Marquardt (default, fast for small to medium problems)

% Set training function (backpropagation variant)
net.trainFcn = 'traingdx';

% Default transfer functions:
% Hidden layers use tansig and output layer uses linear.

net.layers{1}.transferFcn = 'logsig';
net.layers{2}.transferFcn = 'logsig';
net.layers{3}.transferFcn = 'logsig';
% net.layers{4}.transferFcn = 'logsig';

% Training-validation split:
net.divideParam.trainRatio = 1.0;
net.divideParam.valRatio   = 0.0;
net.divideParam.testRatio  = 0.0;

net.trainParam.epochs=100000;
net.trainParam.min_grad = 1e-10;

% net = configure(net,Xd,Yd);
net = train(net,Xd,Yd);
```

Chapter 20

Machine Learning: Mathematical Aspects

Overview

In the previous chapter we saw how training a neural network amounts to solving an optimization problem via the method of steepest descent. The computation of the gradients is rendered very efficient by means of back-propagation. In this chapter we look at how the training can be further accelerated using the Stochastic Gradient-Descent Algorithm. We also look at the issue of which functions are well approximated by a neural network.

20.1 Stochastic Gradient Descent

To recap, we recall the loss function introduced in Chapter 19:

$$\widetilde{\mathcal{L}}(\boldsymbol{p}) = \tfrac{1}{n}\sum_{i=1}^{n} \underbrace{\|f(\boldsymbol{x}_i, \boldsymbol{p}) - \boldsymbol{y}_1\|_2^2}_{=C_i(\boldsymbol{p})}. \tag{20.1}$$

To minimize the loss function (20.1), a method of steepest decents was proposed, with derivatives computed via back-propagation:

$$\boldsymbol{p}^{N+1} = \boldsymbol{p}^N - \eta \nabla_p \widetilde{\mathcal{L}}(\boldsymbol{p}^N). \tag{20.2}$$

Now, because the loss function (20.1), is a sum of squares, we can identify:

$$\widetilde{\mathcal{L}}(\boldsymbol{p}) = \tfrac{1}{n}\sum_{i=1}^{n} C(\boldsymbol{p}; \boldsymbol{x}_i, \boldsymbol{y}_i), \tag{20.3}$$

where the functional form f is assumed to be given and the parameters $\boldsymbol{p}$ are unknown. This will also be written as:

$$\widetilde{\mathcal{L}}(\boldsymbol{p}) = \tfrac{1}{n}\sum_{i=1}^{n} C_i(\boldsymbol{p}). \tag{20.4}$$

Thus, the SD method (20.2) becomes:

$$\boldsymbol{p}^{N+1} = \boldsymbol{p}^N - \eta\left[\frac{1}{n}\sum_{i=1}^{n}\nabla_p C_i(\boldsymbol{p}^n)\right]. \tag{20.5}$$

The idea of stochastic gradient descent is to randomly choose an integer between 1 and n (say i), and to replace Equation (20.2) with:

$$\boldsymbol{p}^{N+1} = \boldsymbol{p}^N - \eta\nabla_p C_i(\boldsymbol{p}^N). \tag{20.6}$$

For η sufficiently small, this still leads *on average* to a reduction in the loss function over many iterations.

To see this, take:

$$\begin{aligned}
\widetilde{\mathcal{L}}(\boldsymbol{p}^{N+1}) - \widetilde{\mathcal{L}}(\boldsymbol{p}^N) &= \widetilde{\mathcal{L}}(\boldsymbol{p}^N - \eta\nabla C_i(\boldsymbol{p}^N)) - \widetilde{\mathcal{L}}(\boldsymbol{p}^N),\\
&= \cancel{\widetilde{\mathcal{L}}(\boldsymbol{p}^N)} - \eta\nabla_p\widetilde{\mathcal{L}}(\boldsymbol{p}^N)\cdot\nabla C_i(\boldsymbol{p}^N) - \cancel{\widetilde{\mathcal{L}}(\boldsymbol{p}^N)},\\
&= -\eta\left(\frac{1}{n}\sum_{j=1}^{n}\nabla C_j(\boldsymbol{p}^N)\right)\cdot\nabla C_i(\boldsymbol{p}^N),\\
&= -\frac{\eta}{n}|\nabla C_i(\boldsymbol{p}^N)|^2 - \eta\left(\frac{1}{n}\sum_{j\neq i}^{n}\nabla C_j(\boldsymbol{p}^N)\right)\cdot\nabla C_i(\boldsymbol{p}^N).
\end{aligned}$$

Hence,

$$\begin{aligned}
\widetilde{\mathcal{L}}(\boldsymbol{p}^{N+1}) - \widetilde{\mathcal{L}}(\boldsymbol{p}^N) = -\frac{\eta}{n}|\nabla C_{i_0}(\boldsymbol{p}^0)|^2 - \frac{\eta}{n}|\nabla C_{i_1}(\boldsymbol{p}^1)|^2 - \cdots - \frac{\eta}{n}|\nabla C_{i_n}(\boldsymbol{p}^N)|^2\\
- \eta\underbrace{\sum_{\alpha=1}^{N}\left[\frac{1}{n}\sum_{j\neq i_\alpha}\nabla C_j(\boldsymbol{p}^\alpha)\cdot\nabla C_{i_\alpha}(\boldsymbol{p}^\alpha)\right]}_{},
\end{aligned}$$

where each $i_\alpha \in \{1, 2, \cdots, n\}$ is an integer drawn at random, with replacement. The idea is that the term in the underbrace contains a large number of dot products of random vectors, which are not aligned, and hence, 'should' average to to zero, over a large number of iterations $N \to \infty$). This can be made precise using stochastic differential equations. Notice that the Stochastic Gradient Descent (SGD) does not reduce the cost function from one iteration to the next; the reduction only happens on average, over a large number of iterations.

20.2 SGD with Momentum

To speed up the SGD algorithm, *SGD with momentum* can be used [8]:

$$\boldsymbol{m}^N = \beta \boldsymbol{m}^{N-1} + (1-\beta)\nabla C_i(\boldsymbol{p}^N), \tag{20.7a}$$

$$\boldsymbol{p}^{N+1} = \boldsymbol{p}^N - \eta \boldsymbol{m}^N. \tag{20.7b}$$

where $\boldsymbol{m}^N$ is the momentum, $\beta \in [0,1)$ is a another parameter, and $\boldsymbol{m}^0 = 0$ is the initial momentum. Notice that $\beta = 0$ gives back ordinary SGD.

Remark: Equation (20.7) closely resembles a Hamiltonian system:

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \boldsymbol{m}(t), \qquad \frac{\mathrm{d}\boldsymbol{m}(t)}{\mathrm{d}t} = -\nabla C_i(\boldsymbol{x}(t)),$$

with Hamiltonian

$$H = \tfrac{1}{2}\boldsymbol{m}^2 + C_i(\boldsymbol{x}),$$

hence

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \frac{\partial H}{\partial \boldsymbol{m}}, \qquad \frac{\mathrm{d}\boldsymbol{m}(t)}{\mathrm{d}t} = -\frac{\partial H}{\partial \boldsymbol{x}}.$$

Performing a temporal discretization on these equations gives:

$$\boldsymbol{x}^{N+1} = \boldsymbol{x}^N + \Delta t\, \boldsymbol{m}^N, \qquad \boldsymbol{m}^N = \boldsymbol{m}^{N-1} - \Delta t\, \nabla C_i(\boldsymbol{x}^N).$$

20.3 Universal Approximation Theorem

In this section we seek to approximate the function

$$f(x) = \sin(x) + x\cos(3x) - 0.1\cos(10x) \tag{20.8}$$

using an ANN. The strategy is to use a shallow neural network, with one hidden layer:

- Input layer, x
- Single hidden layer, N neurons, sigmoidal activation functions;
- Output layer with linear activation function.

Then, the compositions described in the previous chapter simplify:

$$y = \underbrace{\sum_{i=1}^{N} w_i^{(2)} \sigma(w_i^{(1)} x + b_i^{(1)})}_{=g_N(x)}. \tag{20.9}$$

Thus, we are approximating $f(x)$ as $g_N(x)$:

$$f(x) \approx g_N(x).$$

Here, we try to make this approximation precise. First, a worked example.

20.3.1 Example

The function (20.8) is plotted in Figure 20.1, on the interval $[-\pi, \pi]$. Next, the function (on the

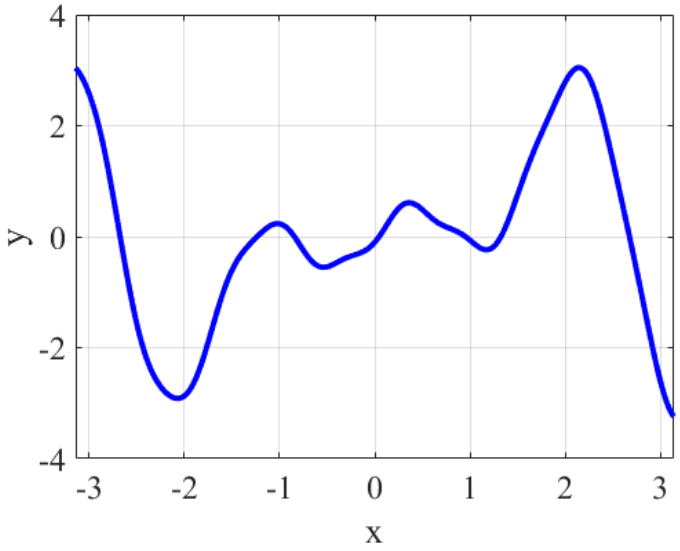


Figure 20.1: Training data for the ANN

same interval) is approximated by a sequence of functions of type $g_N(x)$ (see Equation (20.9)), for various values of N. As N increases, the approximation gets better and better (Figure 20.2). Here, the approximation (20.9) is a simple ANN with a single hidden layer. The biases in the output layer have been forced to be zero. The ANN is trained on the function itself, since this is known – the function is sampled at discrete points $\{x_i\}_{i=1^d}$ to yield values $\{y_i = f(x_i)\}_{i=1}^d$ which provides the training data. Listings for the training of the artificial neural network are provided in Section 21.3. We can also visualize the functions represented by the various trained neurons for $N = 10$ (Figure 20.3).

20.3.2 Universal Approximation Theorem

The previous numerical exercise is made more precise by the so-called **universal approximation theorem**.

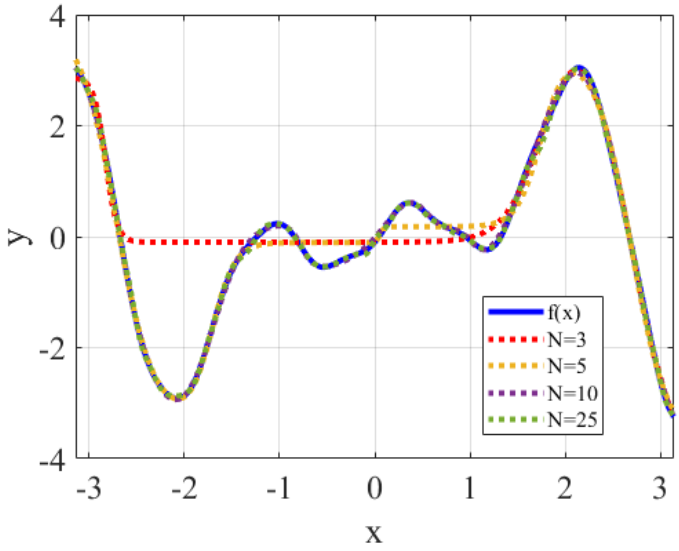


Figure 20.2: Result of fitting an artificial neural network to a function. The training is carried out on $d = 100$ datapoints, sampled evenly in the interval $[-\pi, \pi]$. The value N labels the number of neurons in the hidden layer.

Theorem 20.1 *Let $I = [0, 1]$ and let $f : I \to \mathbb{R}$ be any continuous function on I, equipped with L^∞ norm $\|f\|_\infty = \sup_{x \in I} |f(x)|$. Let $\epsilon > 0$ be given. Then there exists a positive integer N and a function $g_N(x)$ of the form:*

$$g_N(x) = \sum_{i=1}^{N} c_i \sigma(w_i x + b_i), \tag{20.10}$$

such that

$$\|g_N(x) - f(x)\|_\infty < \epsilon. \tag{20.11}$$

The result extends to functions $f : [0, 1]^d \to \mathbb{R}$. The result also extends to arbitrary intervals $[a, b]$, by rescaling. The results also extend to other types of activation function (other than sigmoidal), so in the next chapter we investigate these in more detail.

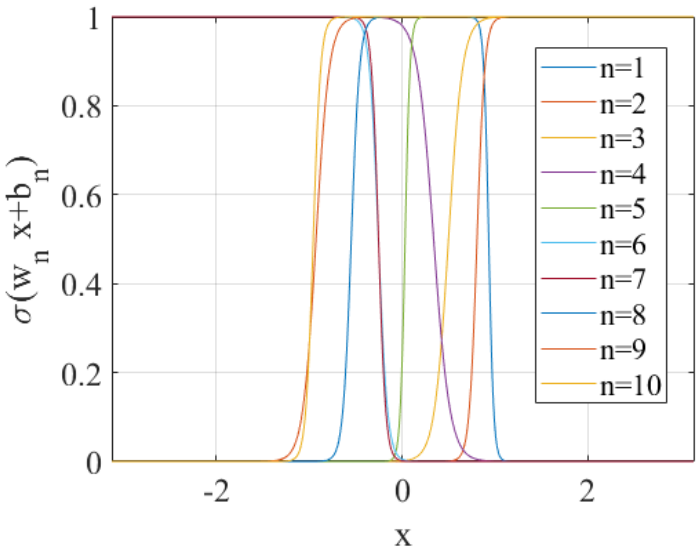


Figure 20.3: Result of fitting an artificial neureal network to a function. The training is carried out on $d = 100$ datapoints, sampled evenly in the interval $[-\pi, \pi]$. The value N labels the number of neurons in the hidden layer.

Chapter 21

Activation Functions; Over-Fitting

Overview

In this chapter we look at other topics in Machine Learning: Activation Functions, and how to avoid Over-Fitting.

21.1 Other Activation Functions

Until now we have assumed that the activation function $\sigma(x)$ is a sigmoidal function. Depending on the application, different activation functions are possible. Here, we go through them all in turn.

21.1.1 Heaviside Step Function

$$H(x) = \begin{cases} 1, & x > 0, \\ 0, & x \leq 0. \end{cases} \tag{21.1}$$

See Figure 21.1.

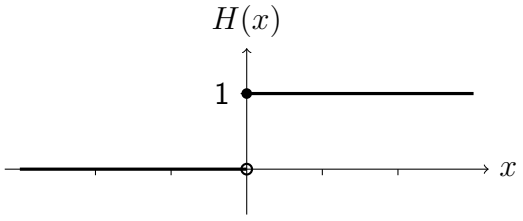


Figure 21.1: The Heaviside Step Function

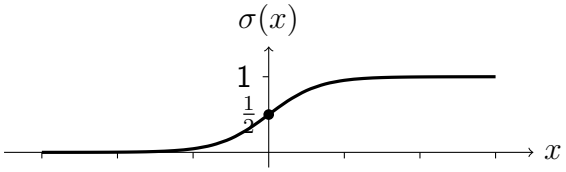


Figure 21.2: Sigmoid activation function

21.1.2 Linear Activation Function

$$\ell(x) = x \tag{21.2}$$

- Pros: Continuous and smooth;
- Cons: As the composition of a linear functions is still linear, the approximating function is linear – back to linear regression!

21.1.3 Sigmoid

The one we have been using so far:

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}}. \tag{21.3}$$

See Figure 21.3.

- Pros: Continuous and smooth;
- Cons: Derivative $\sigma'(x) = \sigma(x)\,[1 - \sigma(x)]$ vanishes far from zero only, derivative has only positive values (creates issues in training).

21.1.4 Tanh

$$\tanh(x) = \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}}. \tag{21.4}$$

See Figure 21.3.

- Pros: Continuous and smooth, outcome is symmetric with respect to zero;
- Cons: Vanishing derivative, for $|x|$ large.

21.1.5 Rectified Linear Unit

$$\mathrm{ReLU}(x) = \begin{cases} x, & x > 0, \\ 0, & x \leq 0. \end{cases} \tag{21.5}$$

See Figure 21.6.

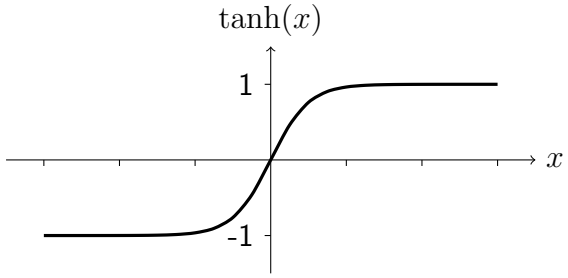


Figure 21.3: Tanh activation function

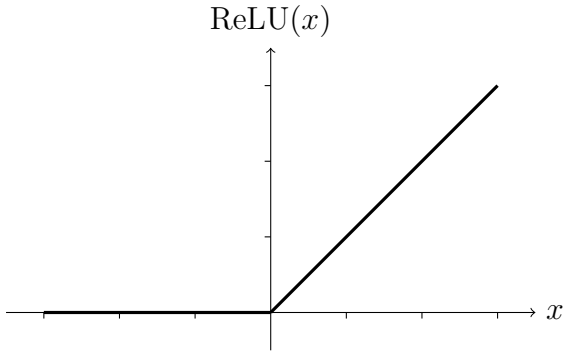


Figure 21.4: ReLU activation function

- Pros: Fast to compute
- Cons: Derivative is zero for negative inputs.

21.1.6 Parametric Rectified Linear Unit

$$\mathrm{PReLU}(x) = \begin{cases} x, & x > 0, \\ \alpha x, & x \leq 0, \end{cases} \tag{21.6}$$

where $\alpha < 1$. See Figure 21.5.

- Pros: No zero gradient
- Cons: Could depend heavily on choice of α.

21.1.7 Swish

$$S(x) = \frac{x}{1 + \mathrm{e}^{-x}} = x\sigma(x). \tag{21.7}$$

See Figure 21.6.

- Pros: Smooth, outperforms ReLU for image classification problems [9].
- Cons: Expensive to compute, non-convex.

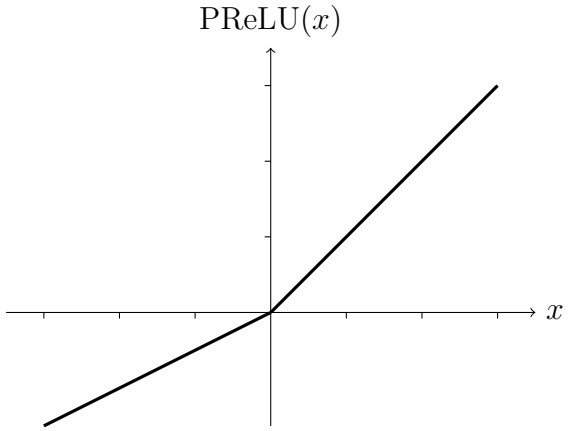


Figure 21.5: Parametrized ReLU activation function

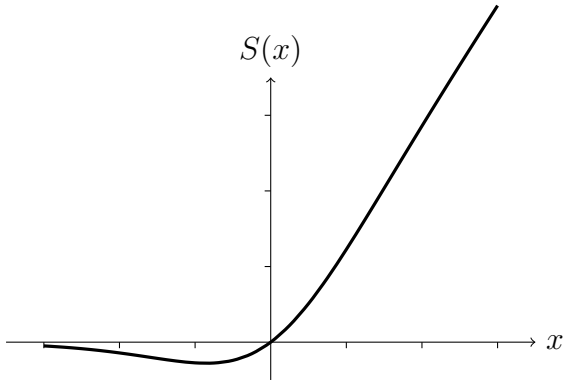


Figure 21.6: Swish activation function

21.2 Avoiding Over-Fitting

Overfitting:

- Network performs very accurately on the given data;
- Network does not generalize well to new data.

Reason – the fitting process has focused too heavily on unimportant and unrepresentative features in the data. Methods for avoiding over-fitting are listed below.

21.2.1 Regularization

In regularization, a penalty is added to the cost function, to penalize over-fitting. Possible penalties include:

- L^2 regularization:

$$R(\boldsymbol{w}) = \lambda R_{L^2}(\boldsymbol{w}), \qquad R_{L^2}(\boldsymbol{w}) = \tfrac{1}{2} \sum_{i,j,\ell} \left[w_{ij}^{(\ell)} \right]^2 \tag{21.8}$$

- L^1 regularization:

$$R(\boldsymbol{w}) = \lambda R_{L^1}(\boldsymbol{w}), \qquad R_{L^1}(\boldsymbol{w}) = \sum_{i,j,\ell} \left| w_{ij}^{(\ell)} \right| \tag{21.9}$$

- Elastic-net regularization:

$$R(\boldsymbol{w}) = \eta R_{L^2}(\boldsymbol{w}) + (1-\eta) R_{L^1}(\boldsymbol{w}) \tag{21.10}$$

Here, η and λ are tunable hyperparameters.

Remark: In L^1 / elastic net regularization, the cost function is no longer differentiable at $w_{ij}^{(\ell)} = 0$. To solve an Optimization Problem using such a cost function, it is necessary to increase the size of the parameter space, and to turn the problem into a constrained Optimization Problem (See Chapter 12 for the idea).

21.2.2 Training / Validation

Another option is to split the given data into two district groups:

- *Training data* is used in the definition of the cost function that defines the optimization problem. Hence, this data drives the process that iteratively updates the weights.

- *Validation data* is not used in the optimization process – it has no effect on how the weights are chosen. Instead, the validation data is used only to judge the performance of the current network. At each step of the optimization, we can evaluate the cost function corresponding to the validation data. This measures how well the current set of weights is performing on unseen data.

Intuitively, overfitting corresponds to the situation where the optimization process is driving down the cost function, but the cost function evaluated on the validation data is no longer decreasing (so the performance on unseen data is not improving). It is therefore reasonable to terminate the training at a stage where no improvement is seen on the validation data.

21.2.3 Dropout

Look at Figure 21.7 showing the architecture of a typical neural network. This is a large, dense (portion) of a network, which may overfit the data. Therefore, at each epoch in the training, each node is removed with probability p. Then, for that epoch, the optimization solver is run on the remaining network. At the next epoch, a different set of neurons is removed (again with probability p), and another round of training is carried out. At the end of the training, when the network is being used to make predictions, each weight and bias is multiplied by a factor of p for use on the full network.

The philosophy behind this method is explained in Reference [10]. The idea that in a network with n neurons, there are 2^n possible sub-networks formed by removing a number of neurons (think: there are 2^n subsets of the set $\{1, 2, \cdots n\}$). In an ideal world we would carry out an ensemble optimization where we train each sub-model on the data and take an ensemble average to compute the best weighs and biases – but this would be an exponentially slow ($O(2^n)$ process). Drop-out speeds up this ensemble averaging.

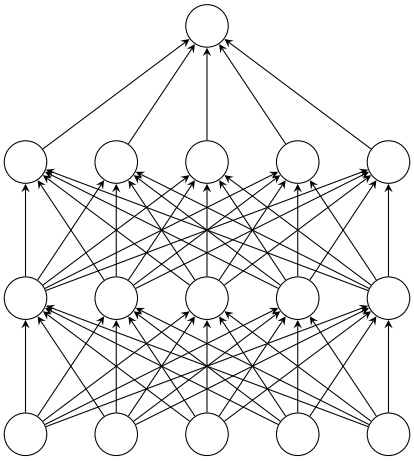


Figure 21.7: Architecture of a portion of a typical neural network

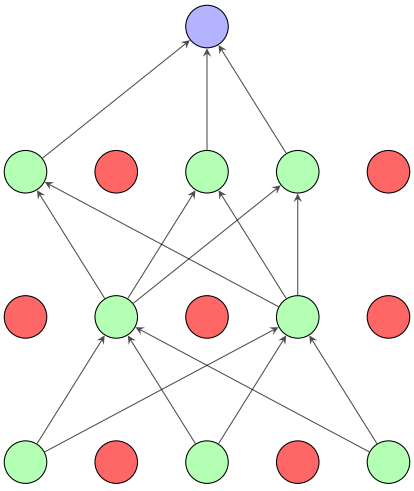


Figure 21.8: The same neural network as in Figure 21.7, but with dropout applied

21.3 Listings

```
function [xdi,y_true,y_interp,b_opt,w1_opt,w2_opt]=shallowNNDemo2(N)

% data points

xd=linspace(-pi,pi,100);
yd=my_fun(xd);

net = feedforwardnet(N);
% Default optimization method:
% Levenberg-Marquardt (default, fast for small to medium problems)

% Set training function (backpropagation variant)
% net.trainFcn = 'traingdx';

% Default transfer functions:
% Hidden layers use tansig and output layer uses linear.

net.layers{1}.transferFcn = 'logsig';
net.layers{2}.transferFcn = 'purelin';

% Zero out output layer bias
net.b{2} = zeros(size(net.b{2}));  % set to zero

% Freeze the output bias so it does not change during training
net.biasConnect(2) = 0;  % disables training of the output layer bias

net = train(net,xd,yd);

b_opt=net.b{1};
w1_opt=net.IW{1,1};
w2_opt=net.LW{2,1};

xdi=linspace(-pi,pi,1000);
y_true=0*xdi;
y_interp=0*xdi;

for ii=1:length(xdi)
    xd_val=xdi(ii);
    y_true(ii)=my_fun(xd_val);
    y_interp(ii)=net(xd_val);
end

end

% ************************************************************************

function y=my_fun(x)
    y=sin(x)+x.*cos(3*x)-0.1*cos(10*x);
end
```

Chapter 22

Convolutional Neural Networks

Overview

Convolutional Neural networks (CNNs) are typically used in image processing problems. They address a fundamental problem in image processing: the input layer is high-dimensional. For example, if the input layer is colour images made up by 200×200 pixels, then this corresponds to an input layer of dimension $200 \times 200 \times 3 = 120,000$ (three for the RGB colour channels in the image). It would be infeasible (and inefficient) to train a regular neural network on such a high-dimensional input layer.

Therefore, CNNs use special weight matrices in the hidden layers to reduce the dimensionality. These matrices are not arbitrary, but are inspired by operations in traditional image processing. We look at these ideas in detail in this chapter, and walk through an example where a CNN is trained on CIFAR-10, a popular database of images.

22.1 Convolution Matrices

Consider a toy problem with an input vector $\boldsymbol{x} = (x_1, \cdots, x_6)^T$ in $\mathbb{R}^6$. Suppose that we multiply $\boldsymbol{x}$ by the matrix

$$W = \begin{pmatrix} 1 & -1 & & & & \\ & 1 & -1 & & & \\ & & 1 & -1 & & \\ & & & 1 & -1 & \end{pmatrix}. \tag{22.1}$$

Thus, we would have:

$$W\boldsymbol{x} = \begin{pmatrix} x_1 - x_2 \\ x_2 - x_3 \\ x_3 - x_4 \\ x_4 - x_5 \\ x_5 - x_6 \end{pmatrix} \in \mathbb{R}^5. \tag{22.2}$$

Therefore, by hitting $\boldsymbol{x}$ with the matrix W, we have reduced the dimension by one. What's more, the entries in the new vector are differences between neighbouring entries in the old vector. In this case we call W **convolution**, obtained by applying the **filter** $[1, -1]$ with stride length 1 – the filter advances one place after each use.

In general, a matrix W is called a **convolution** if its transformation law $\boldsymbol{y} = W\boldsymbol{x}$ can be written in general terms as a convolution:

$$y_k = \sum_{n=1}^{p} x_i g_{k-i}. \tag{22.3}$$

In case of Equation (22.1) we have $g_0 = 1$, $g_-1 = -1$, and all other $g_i = 0$. In the following example:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} a & b & c & d & & & & \\ & & a & b & c & d & & \\ & & & & a & b & c & d \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ 0 \end{pmatrix}, \tag{22.4}$$

the filter is $[a, b, c, d]$, and the stride length is two. Because the dimension of the input vector is not compatible with the filter length, we have **padded** the input vector $\boldsymbol{x}$ with an extra zero. A key observation is that sparse-matrix calculations such as Equation (22.4) can be computed very quickly using fast transform methods, which can be further accelerated using GPUs.

22.1.1 Tensors

In practice, we are applying a convolution to an input image which is effectively three matrices (one $n_x \times n_y$ matrix for each colour channel). This could of course be reshaped into a long column vector but it is more efficient to work with tensors (and the image processing tools all do this anyway). So the input is fixed as a $n_x \times n_y \times 3$ tensor, and the convolution is applied as a small (e.g. 5×5

convolution). However, the convolution mixes colour channels, so the convolution tensor in this case would be $5 \times 5 \times 3$.

In a typical convolutional neural network, there are of course hidden layers. Each layer is spawned by many different convolutions. So we could take an $n_x \times n_y \times 3$ input tensor, and apply N different convolutions each of size $5 \times 5 \times 3$ to generate N different **feature maps**. These feature maps get passed through other convolutions until the final output layer is produced.

22.1.2 Pooling

After (say) taking an input picture of size $n_x \times n_y \times 3$ and generating a hidden layer by convolution with N different $5 \times 5 \times 3$ convolutions (filter width 5) we would have N feature maps, each of size $n_x \times n_y$. So the size of our tensors has ballooned, from $n_x \times n_y \times 3$ to $n_x \times n_y \times N$. In order to reduce the size of the arrays to something more manageable, **pooling** is often applied, e.g. 2×2 squares of neighbouring pixels are replaced by their average value (average pooling) or their maximum value (max pooling). So, if pooling is applied to the present example, each feature map gets reduced in size to $(n_x/2) \times (n_y/2)$.

22.2 Implementation

To demonstrate CNNs, we follow Reference [6] identically. We use the CIFAR-10 dataset, which consists of 50,000 training images (each a $32 \times 32 \times 3 = 3072$ pixel image) that falls into exactly one of the following 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. A further 10,000 images are used for validation. The CNN is trained on the 50,000 training images and then tested on the remaining 10,000 validation images. All data sets can be downloaded from the CIFAR website:

```
https://www.cs.toronto.edu/~kriz/cifar.html
```

The proposed CNN is exactly the one used in Reference [6], and consists of several blocks. **Block 1** consists of a convolution layer which takes in the $32 \times 32 \times 3$ input image and applies a $5 \times 5 \times 3$ filter (the three is for the RGB colour channels, and the of stride is of length 1). A total of 32 different filters are applied, which therefore give rise to 32 different **feature maps**. In this way, the weights can be stored in a $5 \times 5 \times 3 \times 32$ array. Max pooling is then applied to each feature map using stride length two. This reduces the dimension of the feature maps to 16×16. A ReLU activation function is then applied. The idea is shown in Figure 22.1. These steps can be put together as a `block` in Matlab:

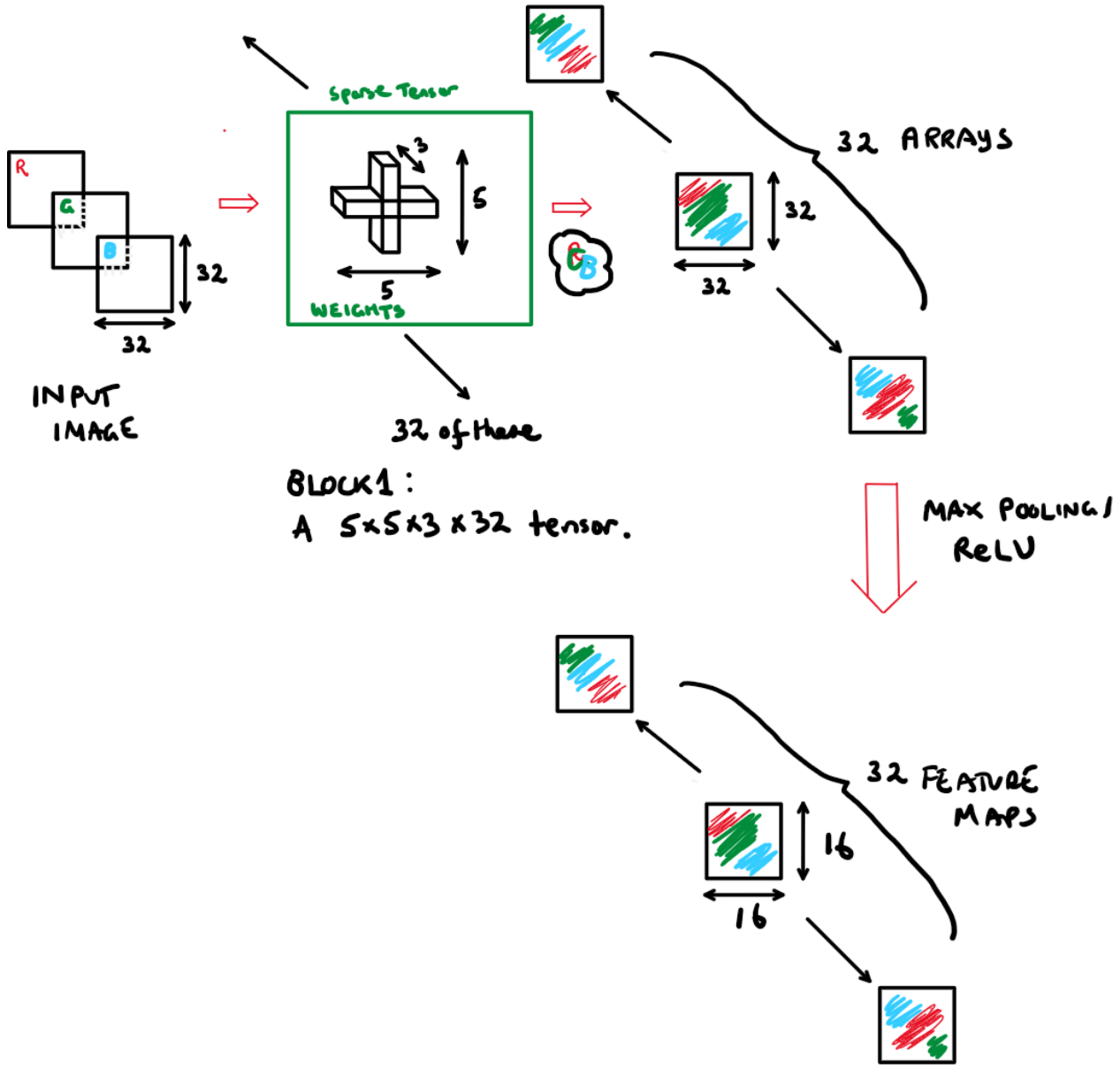


Figure 22.1: Block 1 of the proposed CNN

```
block1 = [
    imageInputLayer([32 32 3], 'Name', 'input')
    convolution2dLayer(5, 32, 'Stride', 1, 'Padding', 'same', 'Name', 'conv1')
    maxPooling2dLayer(2, 'Stride', 2, 'Name', 'pool1')
    reluLayer('Name', 'relu1')
];
```

To further blocks (**block2** and **block 3**) are constructed as follows:

```
block2 = [
    convolution2dLayer(5, 32, 'Stride', 1, 'Padding', 'same', 'Name', 'conv2')  % 16x16x32
    reluLayer('Name', 'relu2')                                                  % 16x16x32
    maxPooling2dLayer(2, 'Stride', 2, 'Name', 'pool2')                          % 8x8x32
];

```

```
block3 = [
    convolution2dLayer(5, 64, 'Stride', 1, 'Padding', 'same', 'Name', 'conv3')
    reluLayer('Name', 'relu3')
    maxPooling2dLayer(2, 'Stride', 2, 'Name', 'pool3') % 8x8x64
];
```

Two final blocks (**block4** and **block 5**) reshape the feature maps into vectors (hence, feature vectors):

- Block 4 uses 64 $4 \times 4 \times 64$ filters, to map 4×4 feature maps into a 64×1 feature vector.
- Block 5 uses a general (fully connected) weight matrix and maps the 64×1 feature vector into a 10×1 feature vector (10 because this is the number of classes in CIFAR-10). The weight matrix is therefore of dimension 10×64.
- A final **softmax** operation transforms each of the ten output components x_i to a number in the range $[0, 1]$ via the transformation:

$$x_i \mapsto \frac{\mathrm{e}^{x_i}}{\sum_{j=1}^{1} 0\mathrm{e}_j^x}. \tag{22.5}$$

The transformation from Block 3 to Block 4 is shown in Figure 22.2. Full details of Blocks 3–5 are

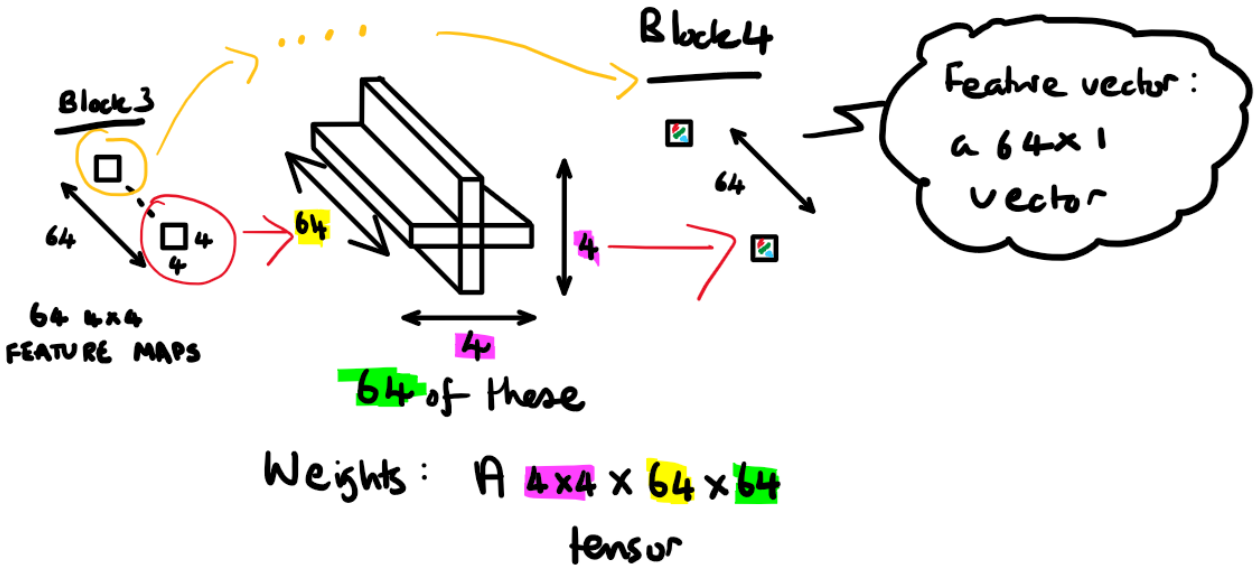


Figure 22.2: Blocks 3 and 4 of the proposed CNN

given in the following listings:

```
fcLayers = [
    fullyConnectedLayer(64, 'Name', 'fc1') % Block 4
    reluLayer('Name', 'relu4')
    dropoutLayer(0.5, 'Name', 'dropout1')   % OPTION: Dropout added here
    fullyConnectedLayer(10, 'Name', 'fc_out') % Block 5
```

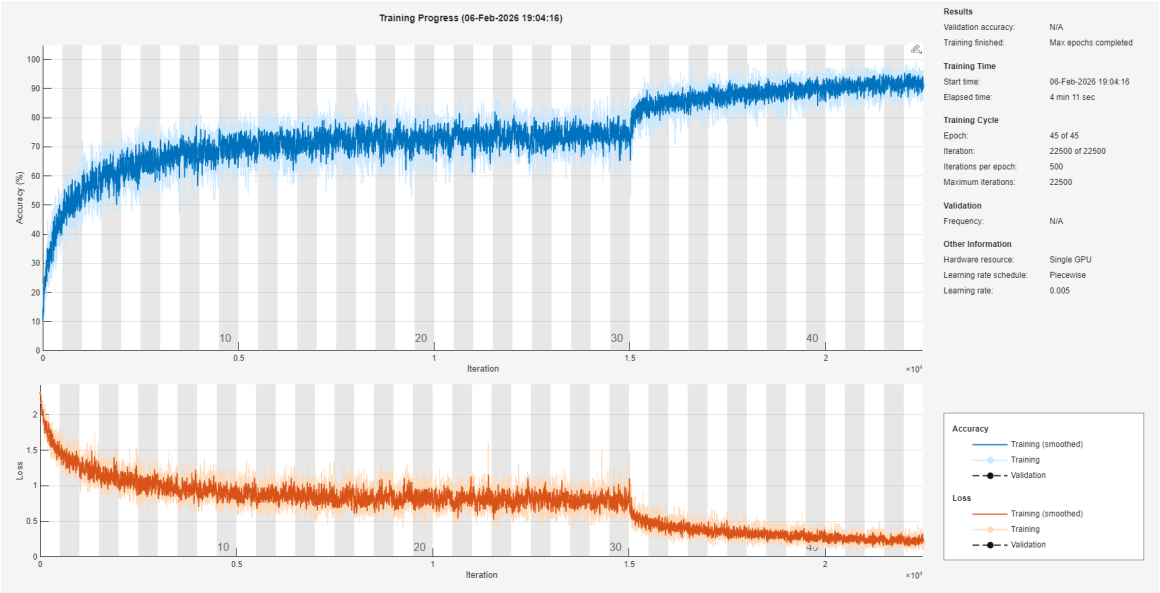


Figure 22.3: Reduction of the cost function with each epoch of training

```
    softmaxLayer('Name', 'softmax') % Output layer uses softmax transformation.
    classificationLayer('Name', 'classoutput')
];
```

22.3 Results

We follow Reference [6], and train the CNN using 50,000 training data and 45 epochs. We apply dropout on Block 4. Stochastic Gradient Descent is used for the optimization step, with intermediate outputs shown in Figure 22.3. The training took just over 4 minutes using a 12-core AMD Ryzen 9 CPU (max clock speed: $4.4\,\mathrm{GHz}$), with 192 GB RAM. The machine was also equipped with an NVIDIA GeForce RTX 3050 GPU and an AMD Radeon GPU. However, the training used the CPU for much of the time. When validated with respect to the 10,000 images in the test set, the trained CNN performed with 74% accuracy (no dropout) and 76% accuracy (with dropout on Block 4).

A **confusion matrix** summarizes the performance of the CNN on the test data. The confusion matrix for the case with dropout is shown in Figure 22.4. Here, the integer value in the (i, j) entry shows the number of occasions where the network predicted category j (predicted) for an image in category i (true). Hence, off-diagonal elements indicate mis-classifications. For example, the $(1, 1)$ entry equal to 804 in Figure 22.4 shows the number of airplane images that were correctly predicted by the CNN, and the $(1, 2)$ shows the 14 times where an airplane image was mis-classified as an automobile image.

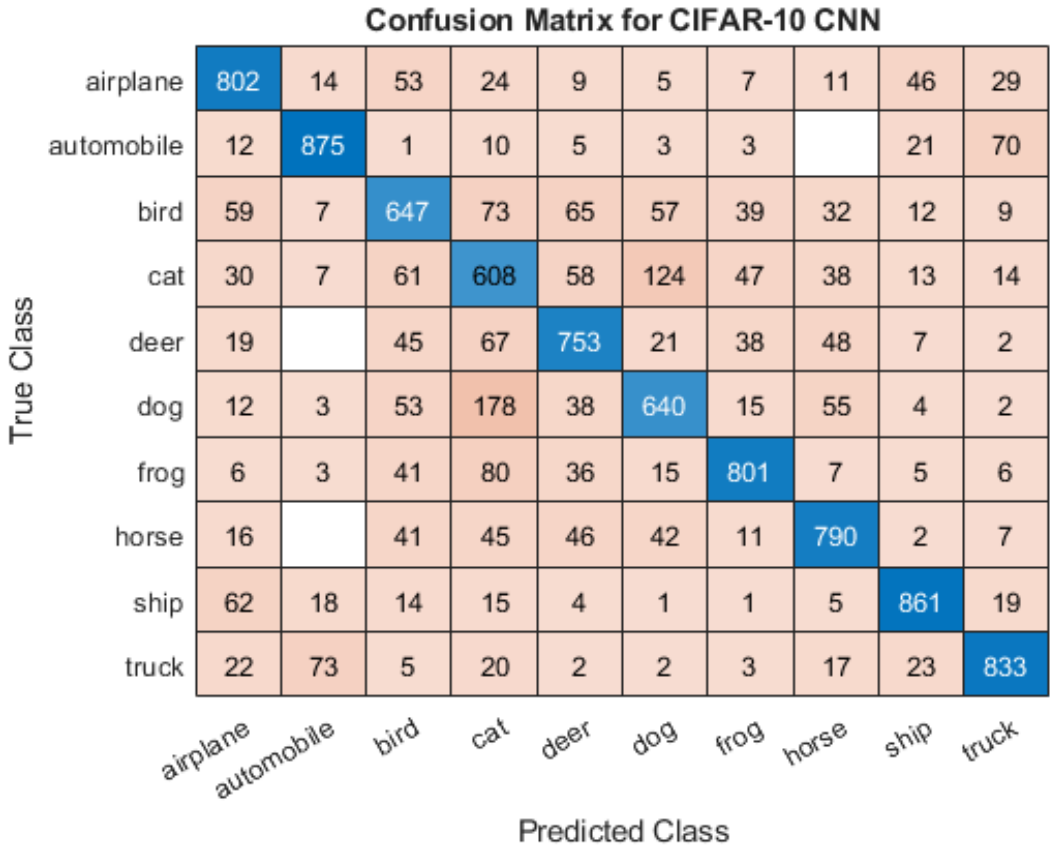


Figure 22.4: Confusion matrix (with dropout)

Chapter 23

Introduction to Support Vector Machines

Overview

We look at Support Vector Machines (SVMs) – a particularly nice way to do binary classification on high-dimensional data sets. The method has a nice geometric interpretation and requires constrained optimization to train the parameters.

23.1 Terminology

Support Vector Machines are well suited to problems of binary classification, where we have a training set of input vectors $\{\boldsymbol{x}_i\}_{i=1^m}$ in some high-dimensional space, $\boldsymbol{x}_i \in \mathbb{R}^D$, for each $i \in \{1, 2, \cdots, m\}$. Suppose that each $\boldsymbol{x}_i$ has a label y_i that is either $+1$ or -1 (or red or blue) (Figure 23.1). A

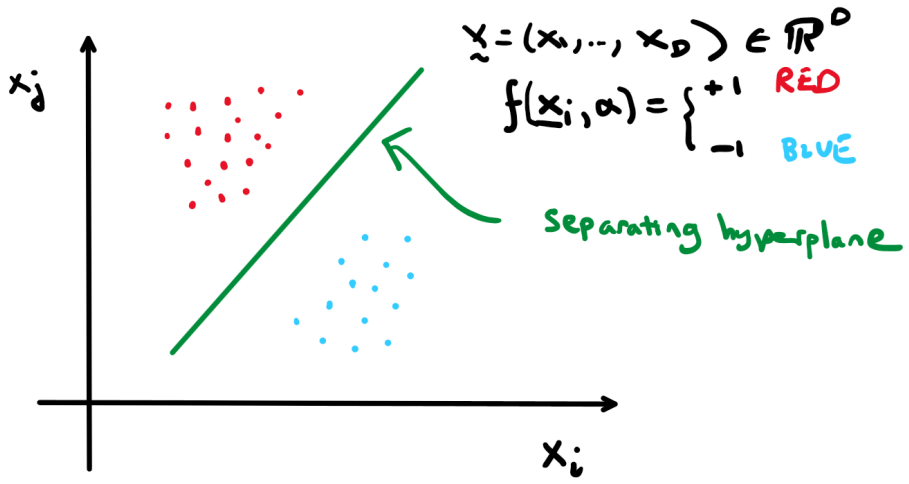


Figure 23.1: The idea behind the binary classification

machine family is a set of functions

$$\{f(\boldsymbol{x},\alpha)\}_{\alpha\in\mathcal{A}}, \qquad \text{such that} f(\boldsymbol{x}_i,\alpha)=y_i, \qquad \text{for each } i\in\{1,2,...,m\}. \tag{23.1}$$

The aim here is to find the 'best' machine in some sense. To quantify 'best', we will have to formulate an optimization problem.

23.2 The Convex Hull Picture

Let

$$A=\{\boldsymbol{x}_i|y_i=1\}, \qquad B=\{\boldsymbol{x}_i|y_i=-1\}. \tag{23.2}$$

Correspondingly, let I_A denote that set of indices such that

$$i\in I_A \text{ if and only if } \boldsymbol{x}_i\in A, \tag{23.3}$$

and similarly for I_B. We introduce the **convex hull** of A and B:

$$\text{conv}(A) = \{\boldsymbol{x}\in\mathbb{R}^D|\boldsymbol{x}=\sum_{i\in I_A}^{m}\alpha_i\boldsymbol{x}_i\}, \tag{23.4a}$$

$$\text{conv}(B) = \{\boldsymbol{x}\in\mathbb{R}^D|\boldsymbol{x}=\sum_{i\in I_B}^{m}\beta_i\boldsymbol{x}_i\}, \tag{23.4b}$$

where the weights α_i and β_i are necessarily non-negative and sum to 1:

$$\sum_{i\in I_A}\alpha_i = 1, \qquad \alpha_i\geq 0, \tag{23.4c}$$

$$\sum_{i\in I_B}\beta_i = 1, \qquad \beta_i\geq 0, \tag{23.4d}$$

For the idea, see Figure 23.2.

23.2.1 Key Assumption

By construction $\text{conv}(A)$ and $\text{conv}(B)$ are non-empty convex subsets of $\mathbb{R}^D$. Therefore, if $\text{conv}(A)$ and $\text{conv}(B)$ are non-overlapping (Figure 23.3), by the separating hyperplane theorem, there exists a non-zero vector $\boldsymbol{w}$ and a constant c such that:

$$\langle\boldsymbol{x},\boldsymbol{w}\rangle\geq c \qquad \text{for all } \boldsymbol{x}\in\text{conv}(A), \tag{23.5a}$$

$$\langle\boldsymbol{x},\boldsymbol{w}\rangle\leq c \qquad \text{for all } \boldsymbol{x}\in\text{conv}(B), \tag{23.5b}$$

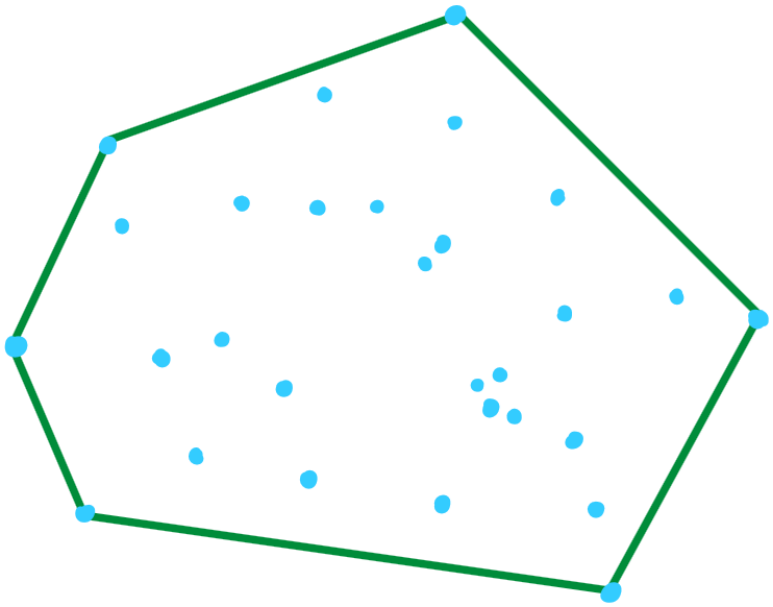


Figure 23.2: The convex hull of a set of points

23.2.2 Equation of a plane

There are different ways of specifying the equation of a plane Π in $\mathbb{R}^n$:

1. A point on the plane $\boldsymbol{r}_0$ and a normal vector $\boldsymbol{n}$:

$$\Pi = \{\boldsymbol{x} \in \mathbb{R}^n | (\boldsymbol{x} - \boldsymbol{r}_0) \cdot \boldsymbol{n} = 0\};$$

2. A normal vector $\boldsymbol{n}$ and a parameter b:

$$\Pi = \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x} \cdot \boldsymbol{n} + b = 0\};$$

3. A normal vector $\boldsymbol{n}$ and a different parameter c:

$$\Pi = \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x} \cdot \boldsymbol{n} = c\}.$$

Obviously, these three approaches are equivalent, if we take:

$$b = \boldsymbol{r}_0 \cdot \boldsymbol{n} = -c, \tag{23.6}$$

and we will use all three approaches interchangeably, whichever one is easier at the time. Here also, we have used the dot product notation momentarily, as this better highlights the geometric interpretation of the equation of the plane.

23.2.3 The OP

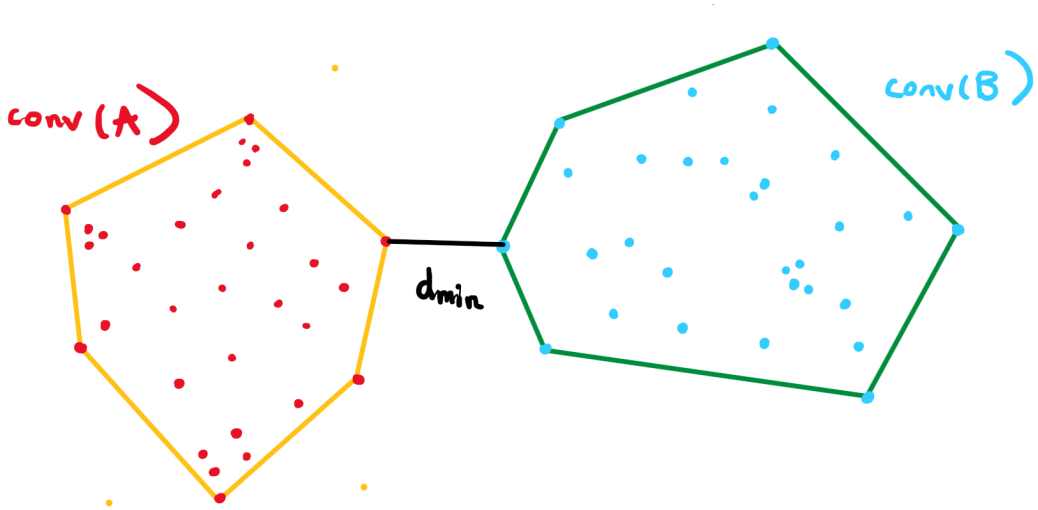


Figure 23.3: $\text{conv}(A)$ and $\text{conv}(B)$ are non-overlapping.

To make progress, we assume in the first instance that $\text{conv}(A)$ and $\text{conv}(B)$ are non-overlapping. We compute the minimum distance between the two convex hulls:

$$d_{min}^2 = \min_{\substack{\boldsymbol{a}\in\text{conv}(A)\\ \boldsymbol{b}\in\text{conv}(B)}} \|\boldsymbol{a}-\boldsymbol{b}\|_2^2$$

This can be written as an optimization problem with constraints:

$$d_{min}^2 = \min_{\alpha_i,\beta_i} \|\sum_{i\in I_A}\alpha_i\boldsymbol{x}_i - \sum_{i\in I_B}\beta_i\boldsymbol{x}_i\|_2^2,$$

subject to:

$$\begin{cases} \alpha_i \geq 0, & \sum_{i\in I_A}\alpha_i = 1, \\ \beta_i \geq 0, & \sum_{i\in I_B}\beta_i = 1. \end{cases} \tag{23.7}$$

Let:

$$\boldsymbol{\lambda} = \{\alpha_1, \cdots; \beta_1, \cdots\}.$$

Thus, the cost function can be re-written as:

$$d_{min}^2 = \min_{\lambda_i\geq 0} \|\sum_{i=1}^{m}\lambda_i y_i \boldsymbol{x}_i\|_2^2,$$

and the constraints $\sum_i \alpha_i = 1$ and $\sum_i \beta_i = 1$ can be re-written as:

$$\sum_{i=1}^{m} y_i\lambda_i = 0.$$

Our constraint optimization problem is therefore:

$$\text{minimize } f(\boldsymbol{\lambda}) = \tfrac{1}{2}\| \sum_{i=1}^{m} \lambda_i y_i \boldsymbol{x}_i \|_2^2, \tag{23.8a}$$

subject to:

$$\lambda_i \geq 0, \qquad \sum_{i=1}^{m} y_i \lambda_i = 0. \tag{23.8b}$$

We expand out the distance function using dot products to obtain the final version of the OP:

$$\text{minimize } f(\boldsymbol{\lambda}) = \tfrac{1}{2} \sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle, \tag{23.9a}$$

subject to:

$$\lambda_i \geq 0, \qquad \sum_{i=1}^{m} y_i \lambda_i = 0. \tag{23.9b}$$

23.2.4 Graphical solution

Given the Separating Hyperplane Theorem, there is always a hyperplane separating $\mathrm{conv}(A)$ and $\mathrm{conv}(B)$. The question we are addressing is, 'what is the optimum separating hyperplane?'. The solution to the OP (1.2) provides the answer. Graphically, the solution can be obtained very easily (but with a little bit of care):

- Case 1. The minimum distance is attained between two vertices of the polygons $\partial\left[\mathrm{conv}(A)\right]$ and $\partial\left[\mathrm{conv}(B)\right]$ (e.g. Figure 23.3)):

  The minimum distance between $\mathrm{conv}(A)$ and $\mathrm{conv}(B)$ is the length of a vector going from a particular vertex on $\mathrm{conv}(A)$ to a particular vertex on $\mathrm{conv}(B)$. Call these vectors $\boldsymbol{x}_a \in A$ and $\boldsymbol{x}_b \in B$. Thus, $d_{min} = \|\boldsymbol{x}_a - \boldsymbol{x}_b\|_2$. The normal to the optimum separating hyperplane is therefore $\boldsymbol{w} = \boldsymbol{x}_a - \boldsymbol{x}_b$. A point on the plane is $\boldsymbol{r}_0 = (\boldsymbol{x}_a + \boldsymbol{x}_b)/2$. Therefore, the optimum separating hyperplane is

  $$\Pi_0 = \{\boldsymbol{x} \in \mathbb{R}^n | (\boldsymbol{x} - \boldsymbol{r}_0) \cdot \boldsymbol{w} = 0\}. \tag{23.10}$$

- Case 2. The minimum distance is attained between a vertex (say on $\partial[\mathrm{conv}(A)]$) and a point on a face of $\partial[\mathrm{conv}(B)]$ (e.g. Figure 23.4)):

  The same procedure as before applies, only now $\boldsymbol{x}_b$ is a linear combination of points $\{\boldsymbol{x}_i\}_{i \in I_B}$.

- Other similar cases...

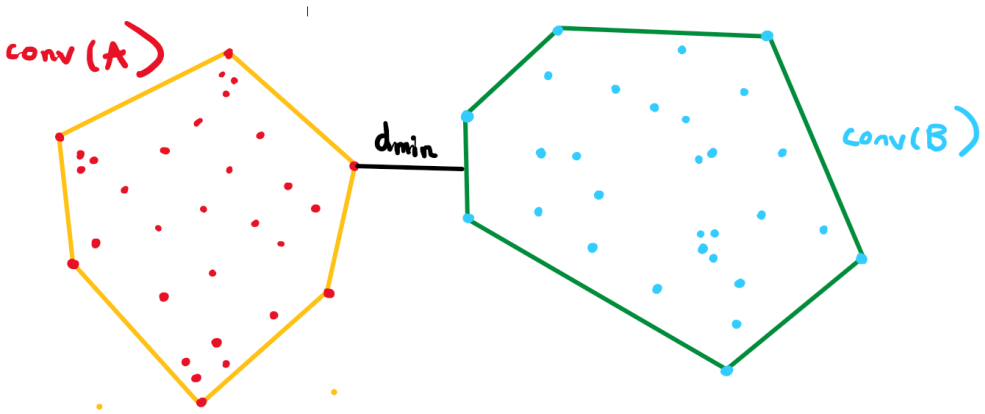


Figure 23.4: The minimum distance is attained between a vertex on $\partial[\mathrm{conv}(A)]$ and a point on a face of $\partial[\mathrm{conv}(B)]$.

23.3 Another approach

Summarizing, the problem we have solved is to construct the best hyperplane separating $\mathrm{conv}(A)$ and $\mathrm{conv}(B)$. In this section we take another approach to the problem. As before, we take any separating hyperplane to be

$$\Pi = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{x}, \boldsymbol{w} \rangle + b = 0\}. \tag{23.11}$$

We have:

$$\langle \boldsymbol{x}, \boldsymbol{w} \rangle + b \geq 0, \qquad \boldsymbol{x}_i \in A, \tag{23.12a}$$

$$\langle \boldsymbol{x}, \boldsymbol{w} \rangle + b < 0, \qquad \boldsymbol{x}_i \in B. \tag{23.12b}$$

We choose a particular scaling for the normal vector $\boldsymbol{w}$ such that:

$$(\boldsymbol{x}_i - \boldsymbol{r}_0) \cdot \boldsymbol{w} \geq 1, \text{ for each } \boldsymbol{x}_i \in A, \tag{23.13}$$

and such that

$$(\boldsymbol{x}_a - \boldsymbol{r}_0) \cdot \boldsymbol{w} = 1 \tag{23.14}$$

where $\boldsymbol{x}_a \in A$ is the point in A closest to the hyperplane.

We further choose the hyperplane to be at the midpoint between the sets A and B, which fixes:

$$(\boldsymbol{x}_i - \boldsymbol{r}_0) \cdot \boldsymbol{w} \leq -1, \text{ for each } \boldsymbol{x}_i \in B, \tag{23.15}$$

and

$$(\boldsymbol{x}_b - \boldsymbol{r}_0) \cdot \boldsymbol{w} = -1. \tag{23.16}$$

The distance between $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ is therefore the projection of $\boldsymbol{x}_a - \boldsymbol{x}_b$ on to the direction normal

to the hyperplane:

$$d = \left| (\boldsymbol{x}_a - \boldsymbol{x}_b) \cdot \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \right|. \tag{23.17}$$

This works out to be:

$$d = \frac{2}{\|\boldsymbol{w}\|}. \tag{23.18}$$

Definition 23.1 *The distance $d = 2/\|\boldsymbol{w}\|$ is called the* **margin**.

To construct the separating hyperplane, we maximize the distance d (best separation), subject to constraints. Equivalently, we may m the vector $\|\boldsymbol{w}\|_2$. Therefore, the constrained optimization problem is:

$$\text{minimize } f(\boldsymbol{w}) = \tfrac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w} \rangle \tag{23.19a}$$

subject to:

$$\begin{aligned} \langle \boldsymbol{x}_i, \boldsymbol{w} \rangle + b \geq 1, \qquad & i \in I_A, \\ \langle \boldsymbol{x}_i, \boldsymbol{w} \rangle + b \leq 1, \qquad & i \in I_B, \end{aligned}$$

Alternatively, we may again use the indicator:

$$y_i = \begin{cases} 1, & \text{if } i \in I_A, \\ -1, & \text{if } i \in I_B. \end{cases} \tag{23.20}$$

Thus, the OP becomes:

$$\text{minimize } f(\boldsymbol{w}) = \tfrac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w} \rangle \tag{23.21a}$$

subject to:

$$y_i \left(\langle \boldsymbol{x}_i, \boldsymbol{w} \rangle + b \right) \geq 1, \qquad i \in I_A \cup I_B.$$

23.3.1 Theoretical Analysis

To understand the constraint OP (23.21) in mathematical terms, we introduce the Lagrangian function

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = f(\boldsymbol{w}) - \sum_{i=1}^{m} \lambda_i \left[y_i \left(\langle \boldsymbol{x}_i, \boldsymbol{w} \rangle + b \right) - 1 \right], \tag{23.22}$$

where $\boldsymbol{\lambda} = (\lambda_1, \cdots, \lambda_m)^T$. The Karush–Kuhn–Tucker conditions (KKT, Chapter 16) give the necessary conditions for first-order optimality:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial b}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = 0, \ \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) &= 0, && \text{(23.23a)} \\ y_i \left(\langle \boldsymbol{x}_i, \boldsymbol{w} \rangle + b\right) - 1 &\geq 0, && \text{(23.23b)} \\ \lambda_i &\geq 0, && \text{(23.23c)} \\ \lambda_i \left[y_i \left(\langle \boldsymbol{x}_i, \boldsymbol{w} \rangle + b\right) - 1\right] &= 0. && \text{(23.23d)} \end{aligned}$$

for each $i \in I_A \cup I_B$.

Only those constraints satisfying

$$y_i \left(\langle \boldsymbol{x}_i, \boldsymbol{w} \rangle + b\right) - 1 = 0, \qquad i \in I_A \cup I_B,$$

have $\lambda_i \neq 0$. The corresponding indices are referred to as the active set, $i \in \mathcal{A}$. These are called the **active constraints**. In Machine-Learning language, the corresponding points $\boldsymbol{x}_i$ are called the **support vectors**. Figure 23.5 shows the idea.

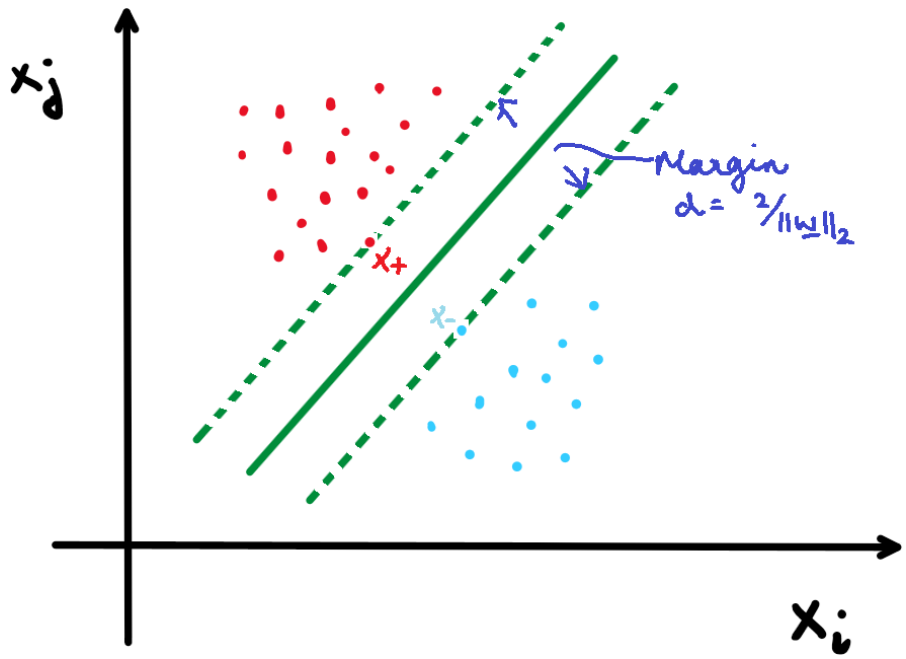


Figure 23.5: Sketch showing the separating hyperplane (solid line), the margins (dashed lines), and the support vectors $\boldsymbol{x}_+$ and $\boldsymbol{x}_-$. Note that there can be more than one support vector on each margin.

Correspondingly, the inactive constraints are far from the separating hyperplane and have:

$$y_i \left(\langle \boldsymbol{x}_i, \boldsymbol{w} \rangle + b\right) - 1 > 0, \qquad i \notin \mathcal{A} \tag{23.24}$$

These **inactive constraints** have $\lambda_i = 0$. The last condition in Equation (23.23) encodes both the active and inactive constraints and is of course the **complementary condition** (see Chapter 16).

23.3.2 Dual Problem

In practice, the OP (23.21) is difficult to solve. However, we may refer to Chapter 17 and solve the **dual problem**. Referring back to the KKT conditions (23.23), first of all compute

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}) = 0. \tag{23.25}$$

This gives

$$\boldsymbol{w} = \sum_{i=1}^{m} \lambda_i y_i \boldsymbol{w}_i. \tag{23.26}$$

Theorem 23.1 *The fact that the optimum vector $\boldsymbol{w}$ is a linear combination of the data points $\boldsymbol{x}_i$ is called the* **Representation Theorem**, *which we hereby get for free, from the KKT conditions.*

Referring back to the KKT conditions (23.23) again, we compute

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda})}{\partial b} = 0. \tag{23.27}$$

This gives

$$\sum_{i=1}^{m} \lambda_i y_i = 0. \tag{23.28}$$

We call:

$$\overline{\boldsymbol{w}} = \sum_{i=1}^{m} \lambda_i y_i \boldsymbol{x}_i. \tag{23.29}$$

We now substitute back into $\mathcal{L}$ to get:

$$\begin{aligned}
\mathcal{L}(\overline{\boldsymbol{w}}, \boldsymbol{\lambda}) &= \tfrac{1}{2}\langle \overline{\boldsymbol{w}}, \overline{\boldsymbol{w}} \rangle - \sum_{i=1}^{m} \lambda_i \left[y_i \left(\langle \overline{\boldsymbol{w}}, \boldsymbol{x}_i \rangle + b \right) - 1 \right], \\
&= \tfrac{1}{2} \sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i=1}^{m} \lambda_i \left[y_i \sum_{j=1}^{m} \lambda_j y_j \left(\langle \boldsymbol{x}_j, \boldsymbol{x}_i \rangle + b \right) - 1 \right], \\
&\overset{\text{Eq. (23.28)}}{=} \tfrac{1}{2} \sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i,j=1}^{m} \lambda_i \lambda_y y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - b \cancel{\sum_{i=1}^{m} \lambda_i y_i} + \sum_{i=1}^{m} \lambda_i, \\
&= -\tfrac{1}{2} \sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \sum_{i=1}^{m} \lambda_i, \\
&= q(\boldsymbol{\lambda}).
\end{aligned}$$

By the Duality Theorem (Chapter 17), the **primal problem** (23.21) is equivalent to solving the **dual problem**

$$\text{maximize } q(\boldsymbol{\lambda}) = -\tfrac{1}{2}\sum_{i,j=1}^{m}\lambda_i\lambda_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle + \sum_{i=1}^{m}\lambda_i, \tag{23.30a}$$

subject to

$$\lambda_i \geq 0, \tag{23.30b}$$

which is equivalent to:

$$\text{minimize } f(\boldsymbol{\lambda}) = \tfrac{1}{2}\sum_{i,j=1}^{m}\lambda_i\lambda_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle - \sum_{i=1}^{m}\lambda_i, \tag{23.31a}$$

subject to

$$\lambda_i \geq 0. \tag{23.31b}$$

Equation (23.31) is the **Hard-Margin Dual Problem**.

23.4 Equivalence of Optimization Problems

If we compare the OP (23.9) (Convex-Hull Optimization Problem) and the OP (23.31) (Hard-Margin Dual Problem), we see they are the same, up to a term

$$\sum_{i=1}^{m}\lambda_i$$

in the Hard-Margin Dual Problem. However, if all the constraints are satisfied, then $\lambda_{i\in I_A} = 1$ and $\lambda_{i\in I_B} = 2$ in the Convex-Hull Optimization problem, meaning that

$$\sum_{i\in I_A, I_B}\lambda_i = 2, \quad \text{Convex-Hull Optimization Problem.}$$

Since the solution to that problem is not changed by adding 2 to the cost function, we conclude that:

Theorem 23.2 *The Convex-Hull Optimization Problem is equivalent to the Hard-Margin Dual Problem.*

Furthermore, since the Hard-Margin Dual problem is equivalent to the primal problem, we see that finding the optimum separating hyperplane can be done in eqivalent ways:

- Finding the shortest distance between the convex hull of A and B and hence constructing the appropriate separating hyperplane;

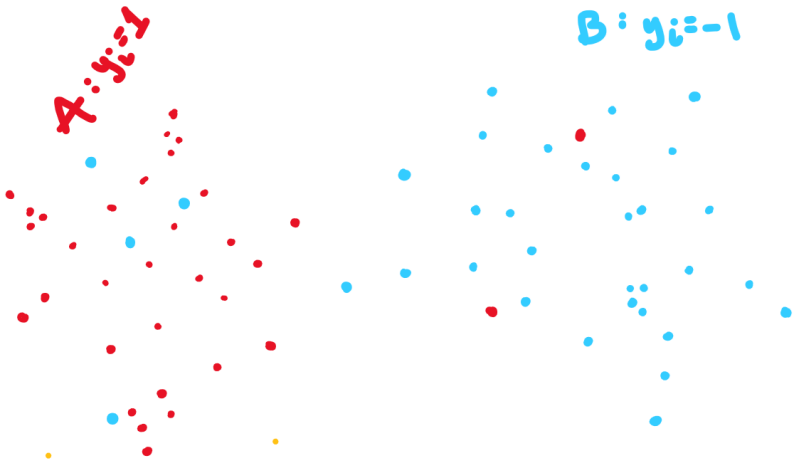


Figure 23.6: Example of A and B not linearly separable

- Maximizing the margin and the support vectors as per OP (23.21).

23.5 Soft Margin

Suppose instead that the data are no longer linearly separable, e.g. Figure 23.6. We consider the reduced convex hulls:

$$R(A,C) = \{\sum_{i\in I_A} \alpha_i \boldsymbol{x}_i | \sum_{i\in I_A} = 1, \quad 0 \leq \alpha_i \leq C\}, \tag{23.32}$$

$$R(B,C) = \{\sum_{i\in I_B} \alpha_i \boldsymbol{x}_i | \sum_{i\in I_B} = 1, \quad 0 \leq \alpha_i \leq C\}. \tag{23.33}$$

We view C as a hyperparameter and compute the minimum distance between the reduced convex hulls.

Caution: We must have $0 < C < 1$.

For the minimum distance, we have:

$$d^2_{min} = \min_{\substack{\boldsymbol{a}\in R(A,C)\\ \boldsymbol{b}\in R(B,C)}} \|\boldsymbol{a}-\boldsymbol{b}\|_2^2. \tag{23.34}$$

This can be re-formulated as:

$$\text{minimize } f(\boldsymbol{\lambda}) = \tfrac{1}{2}\sum_{i,j=1}^{m} \lambda_i\lambda_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle, \tag{23.35a}$$

subject to:

$$0 \leq \lambda_i \leq C, \qquad \sum_{i=1}^{m} y_i \lambda_i = 0. \tag{23.35b}$$

If, instead we formulate the primal problem (similar to Chapter 23)

$$\text{minimize } f(\boldsymbol{w}, \boldsymbol{\xi}) = \tfrac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w} \rangle + C \sum_{i=1}^{m} \xi_m \tag{23.36a}$$

subject to:

$$y_i \left(\langle \boldsymbol{x}_i, \boldsymbol{w} \rangle + b \right) \geq 1 - \xi_i, \qquad i \in I_A \cup I_B,$$

and

$$\xi_i \geq 0, \qquad i \in I_A \cup I_B, \tag{23.36b}$$

we see that the dual problem of Equation (23.36) is equivalent to the OP (23.35). In this context, the ξ_i's are called **slack variables**.

Chapter 24

Reproducing Kernel Hilbert Spaces

Overview

We look at the topic of Reprodudicing Kernel Hilbert Spaces (RKHS) – a fairly abstract concept. Once these are understood, we apply them to classification problems that are not linearly separable, by way of the 'kernel trick'. We build up slowly – starting with finite-dimensional examples, spaces of functions, and then the constrution of a RKHS via a kernel function.

24.1 Motivating Examples

24.1.1 Finite Dimensions

Let $K \in \mathbb{R}^{n \times n}$ be symmetric positive definite, with inverse matrix K^{-1}. Define a weighted inner product,

$$\langle f, g \rangle_K = \langle f, K^{-1} g \rangle_{L^2},$$

where f and g are vectors in $\mathbb{R}^n$ and $\langle \circ, \circ \rangle_{L^2}$ denotes the usual inner product on $\mathbb{R}^n$. Let $\{\boldsymbol{e}_i\}_{i=1}^n$ denote the usual basis on $\mathbb{R}^n$, and define:

$$K_i = K\boldsymbol{e}_i$$

Thus, K_i is a column vector made from the i^{th} column of K, hence $K_i \in \mathbb{R}^n$ also. We have:

$$\begin{aligned} \langle f, K_i \rangle_K &= \langle f, K^{-1} K \boldsymbol{e}_i \rangle_{L^2}, \\ &= \langle f, \boldsymbol{e}_i \rangle_{L^2}, \\ &= f_i. \end{aligned}$$

Thus,

$$\langle f, K_i \rangle_K = f_i,$$

and hence $\langle f, K_i \rangle_K$ evaluates f on the i^{th} coordinate. Furthermore,

$$\begin{aligned} \langle K_i, K_j \rangle_K &= \langle K_i, K^{-1} K \boldsymbol{e}_j \rangle_{L^2}, \\ &= \langle K \boldsymbol{e}_i, \boldsymbol{e}_j \rangle_{L^2}, \\ &= \langle \boldsymbol{e}_i, K \boldsymbol{e}_j \rangle_{L^2}, \\ &= K_{ij}. \end{aligned}$$

Summarizing, the inner product $\langle \circ, \circ \rangle_K$ has the following properties:

- Kernel property:
$$\langle K_i, K_j \rangle_K = K_{ij},$$
- Reproducing property:
$$\langle f, K_i \rangle_K = f_i,$$

These properties make $\mathbb{R}^n$ equipped with $\langle \circ, \circ \rangle_K$ into a **Reproducing Kernel Hilbert Space** (RKHS).

24.1.2 Functional Example

Consider the space $L^2(\mathbb{R})$ of square-integrable functions. Define:

$$K(s, \circ) = \frac{1}{2\alpha} \mathrm{e}^{-|x-\circ|/\alpha}. \tag{24.1}$$

Notice that:

$$(1 - \alpha^2 \partial_x^2) K(s, x) = \delta(x - s).$$

Thus, in some sense, the inverse of the kernel function $K(\circ, \circ)$ is the operator $(1 - \alpha^2 \partial_x^2)$. Therefore, for $\mathcal{H} = H^2(\mathbb{R})$, define the inner product:

$$\langle f, g \rangle_K = \int_{-\infty}^{\infty} f(x)(1 - \alpha^2 \partial_x^2) g(x) \mathrm{d}x, \tag{24.2}$$

Consider:

$$\begin{aligned}\langle f, K(s,\circ)\rangle_K &= \int_{-\infty}^{\infty} f(x)(1-\alpha^2\partial_x^2)K(s,x)\mathrm{d}x,\\ &= \int_{-\infty}^{\infty} f(x)\delta(x-s)\mathrm{d}x,\\ &= f(s).\end{aligned}$$

Also,

$$\begin{aligned}\langle K(s,\circ), K(t,\circ)\rangle_K &= \int_{-\infty}^{\infty} K(s,x)(1-\alpha^2\partial_x^2)K(t,x)\mathrm{d}x,\\ &= \int_{-\infty}^{\infty} K(s,x)\delta(x-t)\mathrm{d}x,\\ &= K(s,t).\end{aligned}$$

Thus, the inner product $\langle\circ,\circ\rangle_K$ has the following properties:

- Kernel property:
$$\langle K(s,\circ), K(t,\circ)\rangle_K = K(s,t).$$
- Reproducing property:
$$\langle f, K(s,\circ)\rangle_K = f(s).$$

These properties make $H^2(\mathbb{R})$ equipped with $\langle\circ,\circ\rangle_K$ into a RKHS.

24.1.3 Fredholm Integral Equations

Consider the Fredholm Integral Equation (FIE):

$$\lambda_i \int K(x,s)\psi_i(s)\mathrm{d}s = \psi_i(s),$$

where $K(x,s)$ is a symmetric **positive definite** kernel. From theory, we know that if $K(x,s)$ is symmetric and continuous, then a set of orthonomal eigenfunctions exist, with

$$\langle\psi_i,\psi_j\rangle_{L^2} = \int_a^b \psi_i(x)\psi_j(x)\mathrm{d}x = \delta_{ij},$$

and with λ_i real. Furthermore, the eigenfunctions span the space

$$\mathcal{H} = \{f \in L^2([a,b]) | f = K * g, \text{some } g \in L^2([a,b])\}.$$

Before introducing the RKHS sturcture, we re-write the kernel in the eigenfunction expansion:

$$K(s, \circ) = \sum_{i=1}^{\infty} k_i(s)\psi_i.$$

In other words,

$$K(s, x) = \sum_{i=1}^{\infty} k_i(s)\psi_i(x), \qquad \text{for all } x \in [a, b].$$

We have:

$$\langle \psi_j, K(s, \circ)\rangle_{L^2} = \sum_{i=1}^{\infty} k_i(s) \underbrace{\langle \psi_j, \psi_i\rangle_{L^2}}_{=\delta_{ij}}.$$

Hence:

$$\frac{1}{\lambda_j}\left[\lambda_j \int_a^b \psi_j(x)K(s, x)\mathrm{d}x\right] = k_j(s).$$

And, since the $\psi_j's$ are eigenvalues of the FIE,

$$\frac{1}{\lambda_j}\psi_j(s) = k_j(s).$$

Thus, the kernel function is represented as:

$$K(s, x) = \sum_{i=1}^{\infty} \frac{\psi_i(s)\psi_i(x)}{\lambda_i}.$$

Remark: By insisting that $K(s, x)$ be positive-definite, we get $\lambda_i > 0$. To see this, start with the positive-definite property:

$$I = \int_a^b \int_a^b K(x, s)f(x)f(s)\mathrm{d}x\mathrm{d}s > 0 \text{ for all } f \in \mathcal{H}.$$

We write f in terms of its eigenfunction expansion:

$$f = \sum_{i=1}^{\infty} f_i\psi_i.$$

We have:

$$\begin{aligned} I &= \int_a^b \int_a^b K(x,s) f(x) f(s) \mathrm{d}x \mathrm{d}s, \\ &= \int_a^b \in_a^b \left(\sum_i \frac{\psi_i(x)\psi_i(s)}{\lambda_i} \right) \left(\sum_j f_j \psi_j(x) \right) \left(\sum_k f_k \psi_k(s) \right) \mathrm{d}s \mathrm{d}x, \\ &= \sum_{ijk} \int_a^b \mathrm{d}s \frac{f_j f_k \psi_i(s)\psi_k(s)}{\lambda_i} \int_a^b \psi_i(x)\psi_j(x)\mathrm{d}x, \\ &= \sum_{ijk} \int_a^b \mathrm{d}s \frac{f_j f_k \psi_i(s)\psi_k(s)}{\lambda_i} \delta_{ij}, \\ &= \sum_{ijk} \frac{f_j f_k}{\lambda_i} \delta_{ij}\delta_{ik}, \\ &= \sum_i \frac{|f_i|^2}{\lambda_i}. \end{aligned}$$

With $I > 0$, we therefore have $\lambda_i > 0$.

Hence, we define the inner product for $\mathcal{H}$. We first take two functions $f, g \in \mathcal{H}$ and write them in terms of their eigenfunction expansions:

$$f = \sum_{i=1}^{\infty} f_i \psi_i, \qquad g = \sum_{i=1}^{\infty} g_i \psi_i.$$

Then,

$$\langle f, g \rangle_K = \sum_{i=1}^{\infty} f_i g_i \lambda_i.$$

Consider:

$$\begin{aligned} \langle f, K(s, \circ) \rangle_{\mathcal{H}} &= \sum_{i=1}^{\infty} f_i \left(\frac{\psi_i(s)}{\lambda_i} \right) \lambda_i, \\ &= \sum_{i=1}^{\infty} f_i \psi_i(s), \\ &= f(s). \end{aligned}$$

Thus, $\langle \circ, \circ \rangle_{\mathcal{H}}$ has the reproducing kernel property.

Furthermore,

$$\begin{aligned}\langle K(x,\circ), K(y,\circ)\rangle_{\mathcal{H}} &= \sum_{i=1}^{\infty} k_i(x)k_i(y)\lambda_i, \\ &= \sum_{i=1}^{\infty} \left(\frac{\psi_i(x)}{\lambda_i}\right)\left(\frac{\psi_i(y)}{\lambda_i}\right)\lambda_i, \\ &= \sum_{i=1}^{\infty} \frac{\psi_i(x)\psi_i(y)}{\lambda_i}, \\ &= K(x,y).\end{aligned}$$

So again, we have:

- Kernel property:
$$\langle K(s,\circ), K(t,\circ)\rangle_K = K(s,t).$$
- Reproducing property:
$$\langle f, K(s,\circ)\rangle_K = f(s).$$

Thus, $\mathcal{H}$ equipped with $\langle\circ,\circ\rangle_K$ is a RKHS.

24.2 General Picture

Let Ω be a generic set of objects. Consider the set of all functions $\mathbb{R}^{\Omega}$, where $f \in \mathbb{R}^{\Omega}$ if and only if,

$$\begin{aligned}f : \Omega &\to \mathbb{R} \\ s &\mapsto f(s).\end{aligned}$$

We assume that all such functions are contained in a space $\mathcal{H}$ equipped with a positive-definite scalar product, making $\mathcal{H}$ into a Hilbert space. We look at the evaluation functional k_s:

$$\begin{aligned}k_s : \mathcal{H} &\to \mathbb{R} \\ f &\mapsto f(s).\end{aligned}$$

The evaluation function k_s takes a function $f \in \mathcal{H}$ and returns a scalar. It is therefore a **functional** on $\mathcal{H}$, with , $k_s[f] = f(s)$. Provided the evaluation function is continuous, by the Riesz Representation Theorem, there exists $K_s \in \mathcal{H}$ such that:

$$\langle f, K_s\rangle = k_s[f] = f(s).$$

Swapping s for t, we have:

$$\langle f, K_t \rangle = f(t).$$

If we simply define:

$$\begin{aligned} K : \Omega \times \Omega &\rightarrow \mathbb{R} \\ (s,t) &\mapsto K(s,t), \end{aligned}$$

where

$$K(s,t) = \langle K_s, K_t \rangle,$$

then we have:

- Kernel property:
$$\langle K_s, K_t \rangle_K = K(s,t).$$
- Reproducing property:
$$\langle f, K_s \rangle_K = f(s).$$

Thus, the evaluation operator makes $\mathcal{H}$ into a RKHS.

24.3 Feature Spaces – Polynomial Kernels

24.3.1 First Example

Using the previous notation, let $\Omega = \mathbb{R}^2$, and define a kernel function

$$\begin{aligned} K : \Omega \times \Omega &\rightarrow \mathbb{R} \\ (\boldsymbol{x}, \boldsymbol{y}) &\mapsto \left(\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{L^2}\right)^2. \end{aligned}$$

Here, $\langle \circ, \circ \rangle_{L^2}$ denotes the usual scalar product on $\mathbb{R}^2$, and:

$$K(\boldsymbol{x}, \boldsymbol{y}) = \left(\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{L^2}\right)^2.$$

Hence:

$$K(\boldsymbol{x}, \boldsymbol{y}) = x_1^2 y_1^2 + 2 x_1 y_1 x_2 y_2 + x_2^2 y_2^2$$

We identify an embedding:

$$\begin{aligned} \phi : \mathbb{R}^2 &\rightarrow \mathcal{H}, \\ \boldsymbol{x} &\mapsto (x_1^2, x_2^2, \sqrt{2} x_1 x_2)^T. \end{aligned}$$

Thus, $\Phi(\boldsymbol{x}) \in \mathbb{R}^3$. We identify $\mathcal{H} = \mathbb{R}^3$, with the usual inner product,

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle_{\mathcal{H}} = \sum_{i=1}^{3} a_i b_i.$$

We have:

$$\begin{aligned} \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_{\mathcal{H}} &= \left(x_1^2, x_2^2, \sqrt{2} x_1 2_2 \right) \begin{pmatrix} y_1^2 \\ y_2^2 \\ \sqrt{y_1} y_2 \end{pmatrix}, \\ &= x_1^2 y_1^2 + 2 x_1 y_1 x_2 y_2 + x_2^2 y_2^2, \\ &= K(\boldsymbol{x}, \boldsymbol{y}) \end{aligned}$$

Furthermore, let $f \in \mathbb{R}^3 = (f_1, f_2, f_3)^T$. We have:

$$\langle f, \phi(\boldsymbol{x}) \rangle_{\mathcal{H}} = f_1 x_1^2 + f_2 x_2^2 + f_3 \sqrt{2} x_1 x_2.$$

If we recognize f as a linear function on $\mathcal{H}$, sending $\mathcal{H}$ to $\mathbb{R}$ via $f : \boldsymbol{a} \mapsto \langle f, \boldsymbol{a}_{\mathcal{H}}$, we have:

$$\langle f, \phi(\boldsymbol{x}) \rangle_{\mathcal{H}} = f(\phi(\boldsymbol{x})).$$

Furthermore, we identify $K(\boldsymbol{x}, \circ) = \phi(\boldsymbol{x})$, hence:

- Kernel property:
$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_K = K(\boldsymbol{x}, \boldsymbol{y}),$$
- Reproducing property:
$$\langle f, \phi(\boldsymbol{x}) \rangle_K = f(\phi(\boldsymbol{x})),$$

Thus, $\mathcal{H}$ is a RKHS with kernel $K(\boldsymbol{x}, \circ) = \phi(\boldsymbol{x})$.

24.3.2 Second Example

We extend the previous example slightly by looking at:

$$\begin{aligned} K : \Omega \times \Omega &\rightarrow \mathbb{R} \\ (\boldsymbol{x}, \boldsymbol{y}) &\mapsto \left(\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{L^2} + b \right)^2. \end{aligned}$$

where b is a parameter. We have:

$$K(\boldsymbol{x}, \boldsymbol{y}) = \left(\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{L^2} + b \right)^2.$$

Hence:

$$K(\boldsymbol{x}, \boldsymbol{y}) = x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 x_2 y_1 y_2 + 2b x_1 y_1 + 2b x_2 y_2 + b^2.$$

We identify an embedding:

$$\begin{aligned} \phi : \mathbb{R}^2 &\rightarrow \mathbb{R}^6, \\ \boldsymbol{x} &\mapsto (1, x_1, x_2, x_1^2, x_2^2, x_1, x_2)^T. \end{aligned}$$

We identify $\mathcal{H} = \mathbb{R}^6$, with weighted inner product:

$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_{\mathcal{H}} = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2) \begin{pmatrix} b^2 & & & & & \\ & 2b & & & & \\ & & 2b & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 2 \end{pmatrix} \begin{pmatrix} 1 \\ y_2 \\ y_2 \\ y_1^2 \\ y_2^2 \\ y_1 y_2 \end{pmatrix}.$$

Extend to arbitrary $\boldsymbol{a}$ and $\boldsymbol{b} \in \mathcal{H}$:

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle_{\mathcal{H}} = (a_1, a_2, a_3, a_4, a_5, a_6) \begin{pmatrix} b^2 & & & & & \\ & 2b & & & & \\ & & 2b & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 2 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix},$$

for all $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{H}$.

Now, let $f \in \mathcal{H} = (f_1, f_2, f_3, f_4, f_5, f_6)^T$. We have:

$$\begin{aligned} \langle f, \phi(\boldsymbol{x}) \rangle_{\mathcal{H}} &= f_1 b^2 + f_2 2b x_1 + f_3 2b x_2 + f_4 x_1^2 + f_5 x_2^2 + f_6 x_1 x_2, \\ &= f(\phi(\boldsymbol{x})). \end{aligned}$$

Again, we recognize f as a linear function on $\mathcal{H}$, sending $\mathcal{H}$ to $\mathbb{R}$ via $f : \boldsymbol{a} \mapsto \langle f, \boldsymbol{a} \rangle_{\mathcal{H}}$. Hence:

$$\langle f, \phi(\boldsymbol{x}) \rangle_{\mathcal{H}} = f(\phi(\boldsymbol{x})).$$

We identify $K(\boldsymbol{x}, \circ) = \phi(\boldsymbol{x})$. We therefore have:

- Kernel property:

$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_{\mathcal{H}} = K(\boldsymbol{x}, \boldsymbol{y}),$$

- Reproducing property:

$$\langle f, \phi(\boldsymbol{x})\rangle_{\mathcal{H}} = f(\phi(\boldsymbol{x})),$$

Thus, $\mathcal{H} = \mathbb{R}^6$ is a RKHS with kernel $K(\boldsymbol{x}, \circ) = \phi(\boldsymbol{x})$.

24.3.3 General Case

In general, we may take $\Omega = \mathbb{R}^n$ and look at the embedding

$$\begin{aligned} K : \Omega \times \Omega &\to \mathbb{R} \\ (\boldsymbol{x}, \boldsymbol{y}) &\mapsto \left(\langle \boldsymbol{x}, \boldsymbol{y}\rangle_{L^2} + b\right)^d. \end{aligned}$$

where b and d are, and $\langle \circ, \circ\rangle_{L^2}$ denotes the usual inner product on $\mathbb{R}^n$. We have:

$$K(\boldsymbol{x}, \boldsymbol{y}) = \left(\langle \boldsymbol{x}, \boldsymbol{y}\rangle_{L^2} + b\right)^2.$$

We identify an embedding:

$$\begin{aligned} \phi : \mathbb{R}^n &\to \mathbb{R}^D, \\ \boldsymbol{x} &\mapsto (1, x_1, x_2, \dots,)^T. \end{aligned}$$

We identify $\mathcal{H} = \mathbb{R}^D$, with weighted inner product

$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y})\rangle_{\mathcal{H}} = (1, x_1, x_2, \dots) B \begin{pmatrix} 1 \\ y_1 \\ y_2 \\ \vdots \end{pmatrix}.$$

Extend to arbitrary $\boldsymbol{a}$ and $\boldsymbol{b} \in \mathbb{R}^D$:

$$\langle \boldsymbol{a}, \boldsymbol{b}\rangle_{\mathcal{H}} = (a_1, a_2, \cdots) B \begin{pmatrix} b_1 \\ b_2 \\ \vdots \end{pmatrix},$$

for all $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^D$.

Now, let $f \in \mathbb{R}^D$, with $f = (f_1, f_2, \cdots)^T$. We have:

$$\begin{aligned} \langle f, \phi(\boldsymbol{x}) \rangle_{\mathcal{H}} &= (f_1, f_2, \cdots) B \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \end{pmatrix}, \\ &= f(\phi(\boldsymbol{x})). \end{aligned}$$

Again, we recognize f as a linear function on $\mathcal{H}$, sending $\mathcal{H}$ to $\mathbb{R}$ via $f : \boldsymbol{a} \mapsto \langle f, \boldsymbol{a} \rangle_{\mathcal{H}}$. Hence:

$$\langle f, \phi(\boldsymbol{x}) \rangle_{\mathcal{H}} = f(\phi(\boldsymbol{x})).$$

We identify $K(\boldsymbol{x}, \circ) = \phi(\boldsymbol{x})$. We therefore have:

- Kernel property:
$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_{\mathcal{H}} = K(\boldsymbol{x}, \boldsymbol{y}),$$
- Reproducing property:
$$\langle f, \phi(\boldsymbol{x}) \rangle_{\mathcal{H}} = f(\phi(\boldsymbol{x})),$$

Thus, $\mathcal{H} = \mathbb{R}^6$ is a RKHS with kernel $K(\boldsymbol{x}, \circ) = \phi(\boldsymbol{x})$.

24.4 Feature Spaces – General Formulation

Quite generally, if Ω is a generic set of objects equipped with a kernel,

$$\begin{aligned} K : \Omega \times \Omega &\to \mathbb{R} \\ (s, t) &\mapsto K(s, t), \end{aligned}$$

where $K(s, t)$ is symmetric and positive definite, we may define an embedding

$$\begin{aligned} \phi : \Omega &\to \mathcal{H}, \\ s &\mapsto \phi(s), \end{aligned}$$

where each $\phi(s)$ can be identified with $K(s, \circ)$, in the sense that

$$\begin{aligned} \langle \phi(s), \phi(t) \rangle_{\mathcal{H}} &= K(s, t), \\ \langle f, \phi(s) \rangle_{\mathcal{H}} &= f[\phi(s)], \end{aligned}$$

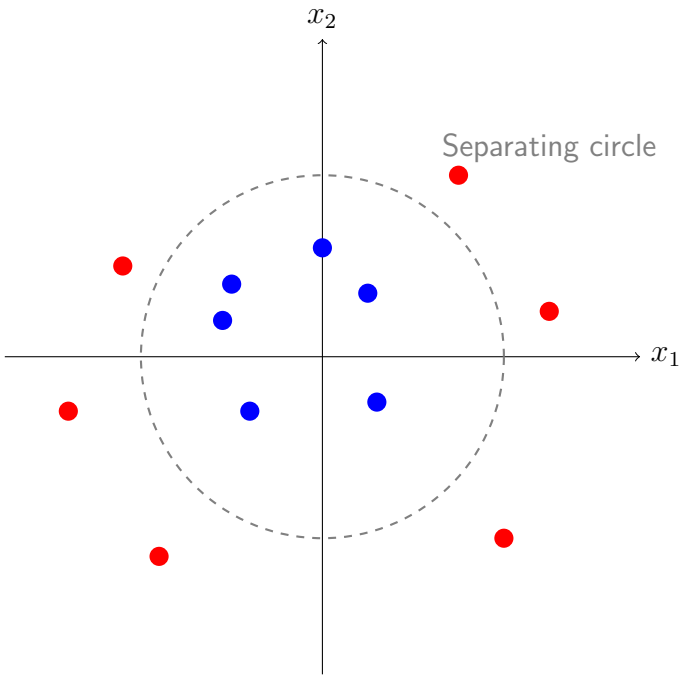


Figure 24.1: Example of non-linearly-separable data in $\Omega = \mathbb{R}^2$

for all s and t in Ω, and all $f \in \mathcal{H}$. The resulting Hilbert space is called a **feature space**.

24.5 Connection to Machine Learning

Often, in a classification problem, the data (examples $\boldsymbol{x}_i$ are not linearly separable). As an alternative to the soft-margin approach, we may consider a feature map $\phi : \mathbb{R}^n \mapsto \mathcal{H}$, such that the mapped date $\phi(\boldsymbol{x}_i)$ are linearly separable in $\mathcal{H}$. For instance, consider the 'circle problem' in Figure 24.1 If we define the feature map

$$\begin{aligned} \phi : \Omega &\rightarrow \mathbb{R}^3, \\ (x_1, x_2) &\mapsto (x_1, x_2, x_1^2 + x_2^2), \end{aligned}$$

then the data are linearly separable in $\mathbb{R}^3$. For $\boldsymbol{z} = (z_1, z_2, z_3)^T \in \mathbb{R}^3$, the separating hyperplane is some $z_3 = r^2$. Thus,

- $z_3 < r^2$, implies $x_1^2 + x_2^2 < r^2$, hence **blue points** in Ω;
- $z_3 > r^2$, implies $x_1^2 + x_2^2 > r^2$, hence **red points** in Ω.

Thus, the classification problem (convex hull description) is solved via the optimization problem:

$$\text{minimize } f(\boldsymbol{\lambda}) = \tfrac{1}{2} \sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle_{\mathbb{R}^3}, \tag{24.3a}$$

subject to:

$$\lambda_i \geq 0, \qquad \sum_{i=1}^{m} y_i \lambda_i = 0. \tag{24.3b}$$

24.5.1 Shortcut

Because each embedding $\phi : \Omega \mapsto \mathcal{H}$ gives rise to a kernel function, and conversely, each kernel function gives rise to an embedding, often it is easier to choose a convenient kernel function, and solve:

$$\text{minimize } f(\boldsymbol{\lambda}) = \tfrac{1}{2} \sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{24.4a}$$

subject to:

$$\lambda_i \geq 0, \qquad \sum_{i=1}^{m} y_i \lambda_i = 0. \tag{24.4b}$$

This can have the advantage that computing kernel functions is computationally fast, whereas computing inner products in the feature space $\mathcal{H}$ can be slow (because the feature space tends to be much higher-dimnensional than Ω). Examples include:

- Polynomial kernel (homogeneous):

$$K(\boldsymbol{x}, \boldsymbol{y}) = \left(\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{L^2}\right)^d,$$

- Polynomial kernel (inhomogeneous):

$$K(\boldsymbol{x}, \boldsymbol{y}) = \left(\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{L^2} + b\right)^d,$$

- Radial basis functions:

$$K(\boldsymbol{x}, \boldsymbol{y}) = \mathrm{e}^{-\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 \gamma}.$$

Bibliography

[1] Stephen Wright, Jorge Nocedal, et al. Numerical optimization. *Springer Science*, 35(67-68):7, 1999.

[2] Lester Ingber. Adaptive simulated annealing (asa): Lessons learned. *Control and Cybernetics*, 25(1):33–54, 1996.

[3] Franz Mandl. *Statistical physics*. John Wiley & Sons, 1991.

[4] Michel Gendreau, Jean-Yves Potvin, et al. *Handbook of metaheuristics*, volume 2. Springer, 2010.

[5] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

[6] Catherine F Higham and Desmond J Higham. Deep learning: An introduction for applied mathematicians. *SIAM review*, 61(4):860–891, 2019.

[7] Charles C Tappert. Who is the father of deep learning? In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 343–348. IEEE, 2019.

[8] Yanli Liu, Yuan Gao, and Wotao Yin. An improved analysis of stochastic gradient descent with momentum. *Advances in Neural Information Processing Systems*, 33:18261–18271, 2020.

[9] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.