LECTURE NOTES

# APPLIED OPTIMIZATION

*Compiled on April 21, 2019*

BENJAMIN PAASSEN,
ANDRÉ ARTELT,
PROF. BARBARA HAMMER

*Faculty of Technology,*
*Bielefeld University*

ACKNOWLEDGEMENTS

Special thanks go to André Artelt, whose supplementary notes on "Introduction to Machine Learning" formed a firm basis and starting point for this document. Thanks for helpful feedback and comments to Arne Kramer-Sunderbrink and André Artelt.

Change log:

**2019-03-26:** Adjusted Definitions 15 and 22 slightly to make handling of continuous cases more intuitive. Also made overall notation more consistent.

**2019-03-29:** Adjusted Definition 22 once more.

**2019-04-21:** Added Section 2.5 on heuristics. Revised the entire script and sorted out many minor mistakes.

# CONTENTS

# INTRODUCTION

This document is meant to accompany the lecture *Applied Optimization* at Bielefeld University. It has been first drafted in summer term 2019 and is based on the excellent supplementary notes on "Introduction to Machine Learning" by André Artelt.

Note that this document deviates slightly from the structure of the lecture in that these notes are structured into two chapters, one on theory and one on algorithms. Further, this preliminary version of the lecture notes does not yet contain meta-heuristics or discrete optimization techniques. Further, the algorithmic chapter covers heuristics – namely the the Downhill-Simplex algorithm, Bayesian Optimization, and CMA-ES – in the same chapter as meta-heuristics.

We emphasize that reading this document is no replacement for visiting the lecture and should rather be regarded as an additional learning resource. Further, this document is likely to contain (hopefully minor) errors and mistakes. If you find any of those, please write to apopt@techfak.uni-bielefeld.de.

# 1 THEORY

This chapter covers excerpts of the theory of optimization. Due to the theoretical topics, the structure is more akin to a math textbook, meaning that we feature a series of definitions, examples, remarks, and theorems. Still, we will try to add intuitive explanations to all concepts. Therefore, if you are uncertain what a definition or a theorem mean, best look for the explanation or example right after it which hopefully makes it more clear.

## 1.1 BASIC CONCEPTS OF OPTIMIZATION

In this section, we first introduce what an optimization problem is, how any optimization problem can be re-written into a certain standard form, and how we can solve some continuous optimization problems analytically by relying on first- and second-order conditions.

### 1.1.1 *Optimization Problems and Formalization*

**Definition 1** (Minimization problem, maximization problem, optimization problem)**.** We define a *minimization problem* as a quartuple consisting of the following elements.

1. a list $\mathcal{X}_1, \ldots, \mathcal{X}_K$ of arbitrary sets, which we call *domains*,
2. a function $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_K \to \mathbb{R}$, which we call *objective function*,
3. a list of tuples $(g_1^l, g_1^r), \ldots, (g_m^l, g_m^r)$, where for all $i$ both $g_i^l : \mathcal{X}_1 \times \ldots \times \mathcal{X}_K \to \mathbb{R}$ and $g_i^r : \mathcal{X}_1 \times \ldots \times \mathcal{X}_K \to \mathbb{R}$ are functions which we call the *left-hand-side* and *right-hand-side* of the $i$th *inequality constraint* respectively, and
4. a list of tuples $(h_1^l, h_1^r), \ldots, (h_n^l, h_n^r)$ where for all $j$ both $h_j^l : \mathcal{X}_1 \times \ldots \times \mathcal{X}_K \to \mathbb{R}$ and $h_j^r : \mathcal{X}_1 \times \ldots \times \mathcal{X}_K \to \mathbb{R}$ are functions which we call the *left-hand-side* and *right-hand-side* of the $j$ *equality constraint* respectively.

We denote a minimization problem as follows.

$$\begin{aligned} \min_{x_1 \in \mathcal{X}_1, \ldots, x_K \in \mathcal{X}_K} \quad & f(x_1, \ldots, x_K) && (1.1) \\ \text{s.t.} \quad & g_i^l(x_1, \ldots, x_K) \geqslant g_i^r(x_1, \ldots, x_K) && \forall i \in \{1, \ldots, m\} \\ & h_j^l(x_1, \ldots, x_K) = h_j^r(x_1, \ldots, x_K) && \forall j \in \{1, \ldots, n\} \end{aligned}$$

We define a *maximization problem* the same way but denote it as follows.

$$\begin{aligned} \max_{x_1 \in \mathcal{X}_1, \ldots, x_K \in \mathcal{X}_K} \quad & f(x_1, \ldots, x_K) && (1.2) \\ \text{s.t.} \quad & g_i^l(x_1, \ldots, x_K) \geqslant g_i^r(x_1, \ldots, x_K) && \forall i \in \{1, \ldots, m\} \\ & h_j^l(x_1, \ldots, x_K) = h_j^r(x_1, \ldots, x_K) && \forall j \in \{1, \ldots, n\} \end{aligned}$$

Both minimization problems and maximization problems are called *optimization problems*.

If $m = n = 0$, we call an optimization problem *unconstrained*.

Intuitively, a minimization problem is concerned with finding values $x_1 \in \mathcal{X}_1, \ldots, x_K \in \mathcal{X}_K$, such that all equality constraints and inequality constraints are fulfilled and such that $f(x_1, \ldots, x_K)$ is as small as possible. Conversely, a maximization problem is concerned with finding values $x_1 \in \mathcal{X}_1, \ldots, x_K \in \mathcal{X}_K$, such that all equality constraints and inequality constraints are fulfilled and such that $f(x_1, \ldots, x_K)$ is as large as possible.

**Remark 2** (Formalization, modelling)**.** Note that it is not always easy to translate an intuitive optimization problem from natural language into the terms of Definition 1. We call this translation process *formalization* or *modelling*. Usually, we may lose information in that translation, because we can not incorporate all the details of real life into our optimization problem. We obtain, instead, a simplified *model* of our actual problem. Then, we obtain a solution for our model, and have to translate this solution back to real life.

To formalize an optimization problem, you can take the following questions as a guideline.

1. How many *variables* do I have? How do I denote them?
2. What is the *domain* for each of those variables?
3. What is the *objective function*, based on these variables?
4. Do I wish to maximize or minimize the objective function?
5. What are *inequality constraints* for my problem?
6. What are *equality constraints* for my problem?

**Example 3** (Optimized tin can)**.** Say you made 1.5 liters of home-made vegan soup for a party and want to transport it to the venue in a can. However, because you are aware of ecological issues you wish to use a can that needs the least amount of material. How does this can need to be shaped?

To model this problem, we follow our recipe above.

1. We first decide to approximate the can with a cylinder. A cylinder can be described by two variables, namely the radius and the height (in centimeters). Let's denote the radius as $r$ and the height as $h$.
2. The domain for both our variables are the real numbers $\mathbb{R}$.
3. Our objective function is the 'material need', which scales with the surface area of the cylinder, including the top and bottom lid. Accordingly, we obtain $f(r, h) = 2\pi \cdot r \cdot h + 2\pi \cdot r^2$.
4. We wish to minimize the surface area.
5. As inequality constraints, we should restrict both variables to be non-negative because a negative radius or height does not make sense. Further, we should ensure that the can has a volume of at least 1.5 liters. The volume of a cylinder is described by the formula $g_1^l(r, h) = \pi \cdot r^2 \cdot h$. Accordingly, we obtain:

$$\begin{aligned} g_1^l(r,h) &= \pi \cdot r^2 \cdot h & g_1^r(r,h) &= 1500 \\ g_2^l(r,h) &= r & g_2^r(r,h) &= 0 \\ g_3^l(r,h) &= h & g_3^r(r,h) &= 0 \end{aligned}$$

6. We have no equality constraints, i.e. $n = 0$.

Our overall minimization problem is thus given as follows.

$$\begin{aligned} \min_{(r,h)\in\mathbb{R}^2} \quad & 2\pi \cdot r \cdot h + 2\pi \cdot r^2 \\ \text{s.t.} \quad & \pi \cdot r^2 \cdot h \geqslant 1500 \\ & r \geqslant 0 \\ & h \geqslant 0 \end{aligned} \tag{1.3}$$

**Example 4** (Minimum cost assignment)**.** Say that we organize a children's birthday party for $K$ children and have packed $K$ bags of sweets with different content. We also have a rough idea about how much each child would like each of the bags and have quantified these preference ratings in a $K \times K$ matrix $\boldsymbol{R}$, where $r_{k,l}$ indicates how much child $k$ would like bag $l$. Now we want to find a one-to-one assignment of bags to children, such that the overall enjoyment is maximized.

1. We can frame our variable as a matrix $\boldsymbol{X}$ where $x_{k,l} = 1$ if and only if we give bag $l$ to child $k$.
2. The domain for our variable is the space of $K \times K$ matrices with binary entries, i.e. $\mathcal{X} = \{0,1\}^{K\times K}$.
3. Our objective function is the 'overall enjoyment', which we can quantify as follows: $f(\boldsymbol{X}) = \sum_{k=1}^{K}\sum_{l=1}^{K} x_{k,l} \cdot r_{k,l}$.
4. We wish to maximize the overall enjoyment.
5. We have no inequality constraints, i.e. $m = 0$.
6. As equality constraints, we wish that every child gets exactly one bag and every bag goes to exactly one child, i.e. we obtain:

$$\begin{aligned} h_1^l(\boldsymbol{X}) &= \sum_{k=1}^{K} x_{k,1} & h_1^r(\boldsymbol{X}) &= 1 \\ & & &\vdots \\ h_K^l(\boldsymbol{X}) &= \sum_{k=1}^{K} x_{k,K} & h_K^r(\boldsymbol{X}) &= 1 \\ h_{K+1}^l(\boldsymbol{X}) &= \sum_{k=1}^{K} x_{1,k} & h_{K+1}^r(\boldsymbol{X}) &= 1 \\ & & &\vdots \\ h_{2\cdot K}^l(\boldsymbol{X}) &= \sum_{k=1}^{K} x_{K,k} & h_{2\cdot K}^r(\boldsymbol{X}) &= 1 \end{aligned}$$

Our overall maximization problem is given as follows.

$$\max_{\boldsymbol{X}\in\{0,1\}^{K\times K}} \quad \sum_{k=1}^{K}\sum_{l=1}^{K} x_{k,l} \cdot r_{k,l} \tag{1.4}$$

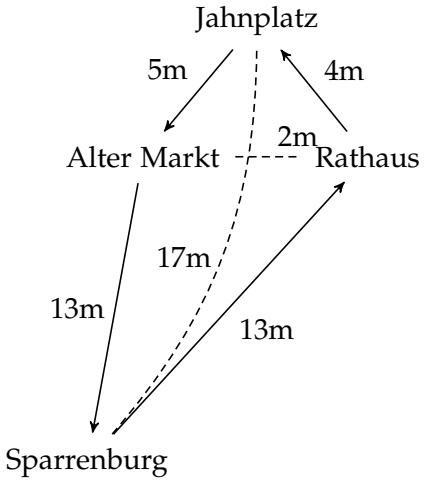



*Figure 1.1:* An instance of the traveling salesperson problem with an optimal round tour, indicated by connecting lines with the travel time in minutes. The travel time for unused connections is annotated with dashed lines

$$\begin{aligned} \text{s.t.} \quad & \sum_{k=1}^{K} x_{k,l} = 1 && \forall l \in \{1, \dots, K\} \\ & \sum_{l=1}^{K} x_{k,l} = 1 && \forall k \in \{1, \dots, K\} \end{aligned}$$

**Example 5** (Traveling salesperson problem)**.** Say that we wish to see all the beautiful sights of Bielefeld downtown, like the Sparrenburg, the old market, the old city hall, the Leineweber memorial, and so on. Overall, there are $m$ sights we wish to visit. However, we also wish to walk as little as possible. Using the internet, we have researched the time we need to walk from each location to each other location and have recorded these times in a $m \times m$ matrix $\boldsymbol{D}$ where $d_{i,j}$ is the time we need from location $i$ to location $j$. Now we wish to find the shortest round trip. Also refer to Figure 1.1.

1. Our variable should be an array $\vec{x}$ with $m$ elements, where $x_t = i$ indicates that we visit location $i$ in the $t$th step of our journey.
2. The domain for our variable is the set of all possible permutations over the set $\{1, \dots, m\}$, which we denote as $\mathcal{X} = \Pi(\{1, \dots, m\})$.
3. Our objective function is the overall time we need for our round trip, which we can quantify as $f(\vec{x}) = \sum_{t=1}^{m-1} d_{x_t, x_{t+1}} + d_{x_m, x_1}$.
4. We wish to minimize the time we need.
5. We have no inequality constraints.
6. We have no equality constraints.

Our overall minimization problem is given as follows.

$$\min_{\vec{x} \in \Pi(\{1, \dots, m\})} \quad \sum_{t=1}^{m-1} d_{x_t, x_{t+1}} + d_{x_m, x_1} \tag{1.5}$$

Note that we could also have formalized this problem differently, e.g. with a matrix variable like in Example 4.

**Remark 6** (Domain and constraints)**.** Note that adding constraints to the problem is equivalent to restricting the domain (also refer to the notion of a feasible set later on). As such, one could argue that constraints are superfluous and we should just specify the domain as a precise set. However, for many practical problems, solutions become much more obvious if we permit a domain that is as general as possible and consider only special kinds of constraints which are easy to handle.

### 1.1.2 *Standard Form*

For our subsequent theory it would be increasingly unwieldy to always distinguish between minimization and maximization problems and consider left- and right-hand sides of constraints separately. Therefore, we introduce the *standard form*, which is considerably simpler but is still expressive enough to capture all possible optimization problems.

**Definition 7** (Standard form)**.** We define an *optimization problem in standard form* as a quartuple with the following ingredients.

1. a single set $\mathcal{X}$, which we call *domain*,
2. a function $f : \mathcal{X} \to \mathbb{R}$, which we call *objective function*,
3. a list $g_1, \ldots, g_m$ of functions $g_i : \mathcal{X} \to \mathbb{R}$, which we call *inequality constraint functions*, and
4. a list $h_1, \ldots, h_n$ of functions $h_j : \mathcal{X} \to \mathbb{R}$, which we call *equality constraint functions*.

We denote an optimization problem in standard form as follows.

$$\begin{aligned} \min_{x \in \mathcal{X}} \quad & f(x) && \\ \text{s.t.} \quad & g_i(x) \geqslant 0 && \forall i \in \{1, \ldots, m\} \\ & h_j(x) = 0 && \forall j \in \{1, \ldots, n\} \end{aligned} \tag{1.6}$$

Now, let

$$\begin{aligned} \min_{x_1 \in \mathcal{X}_1, \ldots, x_K \in \mathcal{X}_K} \quad & f(x_1, \ldots, x_K) && \\ \text{s.t.} \quad & g_i^l(x_1, \ldots, x_K) \geqslant g_i^r(x_1, \ldots, x_K) && \forall i \in \{1, \ldots, m\} \\ & h_j^l(x_1, \ldots, x_K) = h_j^r(x_1, \ldots, x_K) && \forall j \in \{1, \ldots, n\} \end{aligned} \tag{1.7}$$

be a minimization problem and let

$$\begin{aligned} \min_{(x_1, \ldots, x_K) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_K} \quad & \tilde{f}(x) && \\ \text{s.t.} \quad & \tilde{g}_i(x) \geqslant 0 && \forall i \in \{1, \ldots, m\} \\ & \tilde{h}_j(x) = 0 && \forall j \in \{1, \ldots, n\} \end{aligned} \tag{1.8}$$

be an optimization problem in standard form. We call 1.7 and 1.8 *equivalent* if the following conditions hold for all $(x_1, \ldots, x_K), (y_1, \ldots, y_K) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_K$.

$$f(x_1, \ldots, x_K) \geqslant f(y_1, \ldots, y_K) \iff \tilde{f}(x_1, \ldots, x_K) \geqslant \tilde{f}(y_1, \ldots, y_K) \tag{1.9}$$

$$g_i^l(x_1, \ldots, x_K) = g_i^r(x_1, \ldots, x_K) \iff \tilde{g}_i(x_1, \ldots, x_K) = 0 \tag{1.10}$$
$$h_j^l(x_1, \ldots, x_K) = h_j^r(x_1, \ldots, x_K) \iff \tilde{h}_j(x_1, \ldots, x_K) = 0 \tag{1.11}$$

Now, let

$$\begin{aligned} \max_{x_1 \in \mathcal{X}_1, \ldots, x_K \in \mathcal{X}_K} \quad & f(x_1, \ldots, x_K) && \\ \text{s.t.} \quad & g_i^l(x_1, \ldots, x_K) \geqslant g_i^r(x_1, \ldots, x_K) && \forall i \in \{1, \ldots, m\} \\ & h_j^l(x_1, \ldots, x_K) = h_j^r(x_1, \ldots, x_K) && \forall j \in \{1, \ldots, n\} \end{aligned} \tag{1.12}$$

be a maximization problem. We call 1.12 and 1.8 *equivalent* if the following conditions hold for all $(x_1, \ldots, x_K), (y_1, \ldots, y_K) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_K$.

$$f(x_1, \ldots, x_K) \geqslant f(y_1, \ldots, y_K) \iff \tilde{f}(x_1, \ldots, x_K) \leqslant \tilde{f}(y_1, \ldots, y_K) \tag{1.13}$$
$$g_i^l(x_1, \ldots, x_K) = g_i^r(x_1, \ldots, x_K) \iff \tilde{g}_i(x_1, \ldots, x_K) = 0 \tag{1.14}$$
$$h_j^l(x_1, \ldots, x_K) = h_j^r(x_1, \ldots, x_K) \iff \tilde{h}_j(x_1, \ldots, x_K) = 0 \tag{1.15}$$

**Theorem 8** (Universality of standard form)**.** *Let*

$$\begin{aligned} \min / \max_{x_1 \in \mathcal{X}_1, \ldots, x_K \in \mathcal{X}_K} \quad & f(x_1, \ldots, x_K) && \\ s.t. \quad & g_i^l(x_1, \ldots, x_K) \geqslant g_i^r(x_1, \ldots, x_K) && \forall i \in \{1, \ldots, m\} \\ & h_j^l(x_1, \ldots, x_K) = h_j^r(x_1, \ldots, x_K) && \forall j \in \{1, \ldots, n\} \end{aligned} \tag{1.16}$$

*be an optimization problem. Then, the following optimization problem in standard form is equivalent.*

$$\begin{aligned} \min_{(x_1, \ldots, x_K) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_K} \quad & \pm f(x) && \\ s.t. \quad & g_i^l(x) - g_i^r(x) \geqslant 0 && \forall i \in \{1, \ldots, m\} \\ & h_j^l(x) - h_j^r(x) = 0 && \forall j \in \{1, \ldots, n\} \end{aligned} \tag{1.17}$$

*where the objective function is just $f(x)$ if the original problem was a minimization problem and $-f(x)$ if the original problem was a maximization problem.*

*Proof.* First, consider the case where the original problem was a minimization problem. Then, Condition 1.9 is obvious because the objective functions are the same. Conditions 1.10 and 1.11 are obvious by simply subtracting the right-hand side from the inequality/equation.

Second, consider the case where the original problem was a maximization problem. Then, Condition 1.13 follows by multiplying the inequality $f(x) \geqslant f(y)$ with $-1$. Conditions 1.14 and 1.15 are obvious by simply subtracting the right-hand side from the inequality/equation. □

**Remark 9** (Using the standard form)**.** We have motivated the standard form because it makes theoretical arguments easier – whatever we can prove for the standard form automatically applies to all optimization problems. However, the standard form is also useful in practice because most programming libraries for optimization use this standard form or a related form as an input interface. Therefore, being able to convert from any model to the standard form is a useful skill to have.

**Example 10** (Standard forms)**.** The standard form for the tin can problem from Equation 1.3 is as follows.

$$\begin{aligned} \min_{(r,h)\in\mathbb{R}^2} \quad & 2\pi \cdot r \cdot h + 2\pi \cdot r^2 \\ \text{s.t.} \quad & \pi \cdot r^2 \cdot h - 1500 \geqslant 0 \\ & r \geqslant 0 \\ & h \geqslant 0 \end{aligned}$$

The standard form for the minimum cost assignment problem from Equation 1.4 is as follows.

$$\begin{aligned} \min_{\boldsymbol{X}\in\{0,1\}^{K\times K}} \quad & \sum_{k=1}^{K}\sum_{l=1}^{K} x_{k,l} \cdot (-r_{k,l}) & \\ \text{s.t.} \quad & \sum_{k=1}^{K} x_{k,l} - 1 = 0 & \forall l \in \{1,\dots,K\} \\ & \sum_{l=1}^{K} x_{k,l} - 1 = 0 & \forall k \in \{1,\dots,K\} \end{aligned}$$

The traveling Salesperson problem 1.5 is already in standard form.

### 1.1.3 *Global and Local Optima*

Until now we have defined what an optimization problem is, but we have not yet defined what it means to *solve* an optimization problem. We introduce the concept of a solution now.

**Definition 11** (Feasible set, global minimum, solution)**.** We define the *feasible set* $\mathcal{X}^*$ of an optimization problem in standard form as the set

$$\mathcal{X}^* := \left\{ x \in \mathcal{X} \,\middle|\, \forall i \in \{1,\dots m\} : g_i(x) \geqslant 0, \forall j \in \{1,\dots,n\} : h_j(x) = 0 \right\} \tag{1.18}$$

We define a *global minimum* or *solution* of an optimization problem in standard form as an element $x^* \in \mathcal{X}^*$ such that for all $x \in \mathcal{X}^*$ it holds:

$$f(x^*) \leqslant f(x) \tag{1.19}$$

**Example 12** (Logical and)**.** Consider the following optimization problem:

$$\min_{(x,y)\in\{0,1\}^2} -x \cdot y \tag{1.20}$$

The feasible set of this problem is the entire domain, i.e. $\mathcal{X}^* = \{(0,0),(0,1),(1,0),(1,1)\}$. The former three values have an objective function value of 0, the last one a value of $-1$. Accordingly, $(x,y) = (1,1)$ is the global minimum or solution of this problem because there is no other point in the feasible set with a smaller objective function value.

**Remark 13** (Equivalence of solutions)**.** It is straightforward to extend Definition 11 to general optimization problems and to show that any solution of a general optimization problem is also the solution of its equivalent optimization problem in standard form according to Theorem 8 and vice versa. For brevity, we omit this here.

**Remark 14** (Multiple solutions)**.** Note that Definition 11 does permit multiple solutions. As an example, consider the trivial optimization problem

$$\min_{x \in \mathbb{R}} \quad 0$$

All $x \in \mathbb{R}$ are global minima of this problem.

**Definition 15** (Neighborhoods and local minima)**.** Let $\mathcal{X}$ be the domain and $\mathcal{X}^*$ be the feasible set of an optimization problem in standard form. We call a function $\mathcal{N} : \mathcal{X} \to \mathcal{P}(\mathcal{X})$ a *neighborhood function* and we call $\mathcal{N}(x)$ the *neighborhood* of $x$ according to $\mathcal{N}$.

We define a *local minimum* of an optimization problem in standard form with respect to the neighborhood function $\mathcal{N}$ as an element $x^* \in \mathcal{X}^*$ such that it holds:

$$f(x^*) \leqslant f(x) \quad \forall x \in \mathcal{N}(x^*) \cap \mathcal{X}^* \tag{1.21}$$

**Example 16** (Bit-flip neighborhood)**.** Consider the following optimization problem:

$$\min_{(x,y) \in \{0,1\}^2} \quad -x \cdot y$$

And consider the neighborhood function $\mathcal{N}(x, y) := \{(1 - x, y), (x, 1 - y)\}$, i.e. the $x/y$ combinations that result by flipping one bit.

According to that neighborhood function, the global minimum $(1, 1)$ has the neighborhood $\mathcal{N}(1, 1) = \{(0, 1), (1, 0)\}$. Both entries have a larger objective function value, such that the global minimum is also a local minimum according to $\mathcal{N}$.

Now, consider the point $(0, 0)$. The neighborhood is $\mathcal{N}(0, 0) = \{(0, 1), (1, 0)\}$. Both entries have an objective function value of 0, which is the same as the objective function value of $(0, 0)$. Therefore, $(0, 0)$ is also a local minimum according to $\mathcal{N}$, even though it is not a global minimum.

**Example 17** (Swap neighborhood)**.** Consider the Traveling Salesperson-Problem from Equation 1.5 for the distances in Figure 1.1. For this problem, we can define the neighborhood function

$$\mathcal{N}(x_1, x_2, x_3, x_4) = \{(x_2, x_1, x_3, x_4), (x_1, x_3, x_2, x_4), (x_1, x_2, x_4, x_3), (x_4, x_2, x_3, x_1)\}$$

i.e. we swap a neighboring location in the tour.

According to that neighborhood function, the tour (Jahnplatz, Sparrenburg, Alter Markt, Rathaus) is *not* a local minimum, because we can improve it by swapping Sparrenburg and Alter Markt, which yields a global minimum. This global minimum is also a local minimum because any swap will increase our travel time.

**Example 18** (Continuous neighborhood)**.** Consider the optimization problem

$$\min_{x \in \mathbb{R}} \quad x^4 - x^2 + \frac{1}{4}x$$

for which the objective function is shown in Figure 1.2.

We define the neighborhood function

$$\mathcal{N}_\epsilon(x) = \left\{ y \in \mathbb{R} \middle| \, \|x - y\| \leqslant \epsilon \right\} \tag{1.22}$$

i.e. the $\epsilon$-ball around $x$.

For $\epsilon = 0.5$, $x \approx 0.63$ is a local minimum but not a global minimum and $x \approx -0.76$ is both a local minimum and a global minimum.

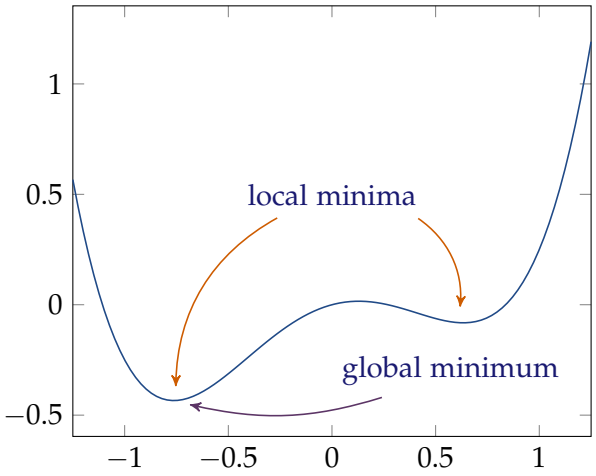



*Figure 1.2:* A plot of the function $f(x) = x^4 - x^2 + \frac{1}{4}x$ with two local minima at $x \approx -0.76$ and $x \approx 0.63$, the former one also being a global minimum.

**Remark 19** (Optimization via neighborhood search)**.** Neighborhood functions directly give us a straightforward tool to solve optimization problems. We simply start at a random point in the feasible set, evaluate the objective function for that point, then compute the neighborhood for the point and evaluate the objective function for all points within the neighborhood. If any of these points has a lower objective function value, we switch to that point. If no neighbor has a better objective function value, we stop. Note that we stop only if we have found a local minimum. Also refer to the optimization method of *hill climbing* (Section 2.5.2).

Consider Example 16 again. If we start off with $x = (1, 0)$, the neighborhood is $\mathcal{N}(1, 0) = \{(0, 0), (1, 1)\}$. We obtain the objective function values $f(1, 0) = 0$, $f(0, 0) = 0$ and $f(1, 1) = -1$. Therefore, we switch to $x = (1, 1)$. All neighbors of $x = (1, 1)$ have a higher objective function value, therefore we stop.

**Theorem 20** (Relationship of local and global optima)**.** *A global minimum is also a local minimum according to every neighborhood function $\mathcal{N}$.*

*Conversely, not every local minimum is a global minimum.*

*Proof.* Regarding the first claim, recall the definition of a global minimum. A global minimum is a point $x^* \in \mathcal{X}^*$ such that $f(x^*) \leqslant f(x)$ for all $x \in \mathcal{X}^*$. Then, in particular, $f(x^*) \leqslant f(x)$ for all $x$ in any subset of $\mathcal{X}^*$. Therefore, irrespective of the neighborhood function, $x^*$ is a local minimum.

The second claim holds because we have provided counterexamples above. □

**Remark 21** (Intuition behind local minima)**.** When we use the term local minima, we typically refer to local minima that are *not* at the same time global minima. These "nasty" local minima are particularly interesting because they can easily mislead an optimization algorithm that is based on neighborhoods. For example, if we apply neighborhood search on Example 16 starting from point $x = (0, 0)$, then we immediately stop, even though we have not found a global minimum yet.

Therefore, solving problems with many local minima that are not global minima is more difficult than solving problems with only a single local minimum which is also the global minimum.

### 1.1.4 *Continuous versus Discrete Optimization Problems*

A fundamental distinction in optimization is between continuous and discrete optimization. Intuitively, continuous optimization is about smooth objective functions and gap-free feasible sets, whereas discrete optimization is concerned with feasible sets with gaps and non-differentiable objective functions with sudden jumps.

Due to these fundamental differences, the optimization strategies we have to employ are fundamentally different as well. In continuous optimization, we can smoothly move through space and observe a continuous change in the objective function value corresponding to our movement. In discrete optimization, we have to perform "jumps" through the feasible set, which we have to control by means of some heuristic.

More formally, we provide the following definition.

**Definition 22** (Continuous function, continuous optimization problem, discrete optimization problem)**.** Let $f$ be a function $f : \mathbb{R}^K \to \mathbb{R}$ for some $K \in \mathbb{N}$. We call $f$ a *continuous function* if for all $\vec{x} \in \mathcal{X}$ and all $\epsilon > 0$ there exists a $\delta > 0$ such that for all $\vec{y} \in \mathcal{N}_\delta(\vec{x})$ it holds $|f(\vec{y}) - f(\vec{x})| < \epsilon$, where $\mathcal{N}_\delta$ refers to the neighborhood in Equation 1.22.

We call an optimization problem *continuous* if

1. the domain is $\mathcal{X} = \mathbb{R}^K$ for some $K \in \mathbb{N}$,
2. the objective function $f$ is a continuous function on the feasible set $\mathcal{X}^*$,
3. the feasible set is *path-connected*, i.e. for any two points $\vec{x}, \vec{y} \in \mathcal{X}^*$ there exists a continuous function $\phi : [0, 1] \to \mathcal{X}^*$, such that $\phi(0) = \vec{x}$ and $\phi(1) = \vec{y}$.

We call an optimization problem *discrete* if the domain $\mathcal{X}$ is countable, i.e. there exists a surjective mapping $\pi : \mathbb{N} \to \mathcal{X}$.

**Example 23** (Continuous and discrete problems)**.** From our previous examples, the tin can problem (Example 3) and Example 18 are continuous and the minimum cost assignment problem (Example 4), the traveling salesperson problem (Example 5), and the logical and problem (Example 1.20) are discrete.

**Remark 24** (Beyond continuous and discrete)**.** Note that there exist optimization problems which are *neither* continuous nor discrete according to Definition 22. Consider the following example.

$$\begin{aligned} \min_{x \in \mathbb{R}} \quad & x^2 \\ \text{s.t.} \quad & |x| \geqslant 1 \end{aligned}$$

The feasible set is $(-\infty, -1] \cup [1, \infty)$, which is neither path-connected - because there is a gap between $-1$ and $1$ - nor is it countable. Therefore, this problem is neither continuous nor discrete.

Some of these more exotic cases can still be addressed with the methods of this course, but we will generally assume that our problems are either discrete or continuous in the sense above.

## 1.2 DIFFERENTIABLE OPTIMIZATION

If the objective function of an optimization problem is continuous and differentiable, but even we can apply the usual optimization machinery that we know from school: Setting the first derivative to zero and checking the second derivative. In this section, we will try to explain why this machinery works and generalize it to multiple dimensions.

### 1.2.1 *Gradient, Hessian, and Taylor Expansion*

The first thing we introduce is a generalization of the first and second derivative to higher dimensions by means of the *gradient* and the *Hessian* respectively.

**Definition 25** (Gradient and Hessian)**.** Let $f$ be a function $f : \mathbb{R}^K \to \mathbb{R}$ for some $K \in \mathbb{N}$. Then, we define the *gradient* of $f$ at position $\vec{x} \in \mathbb{R}^k$ as follows:

$$\nabla_{\vec{x}} f(\vec{x}) = \begin{pmatrix} \frac{\partial}{\partial x_1} f(\vec{x}) \\ \vdots \\ \frac{\partial}{\partial x_K} f(\vec{x}) \end{pmatrix} \tag{1.23}$$

We define the *Hessian* of $f$ at position $\vec{x} \in \mathbb{R}^k$ as follows:

$$\nabla^2_{\vec{x}} f(\vec{x}) = \begin{pmatrix} \frac{\partial^2}{\partial^2 x_1} f(\vec{x}) & \cdots & \frac{\partial^2}{\partial x_1 \partial x_K} f(\vec{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_K \partial x_1} f(\vec{x}) & \cdots & \frac{\partial^2}{\partial^2 x_K} f(\vec{x}) \end{pmatrix} \tag{1.24}$$

The reason why gradient and Hessian are so useful for optimization is because they allow us to indirectly characterize how a objective function behaves without having to regard the precise shape of the objective function. Our central tool in that regard is the *Taylor expansion*.

**Definition 26** (Taylor expansion)**.** Let $f : \mathbb{R}^K \to \mathbb{R}$ be a twice-differentiable function.

We define the first-order Taylor expansion of $f$ around some point $\vec{x}^* \in \mathbb{R}^K$ as the function $\tilde{f}^1_{\vec{x}^*} : \mathbb{R}^K \to \mathbb{R}$ with the following form.

$$\tilde{f}^1_{\vec{x}^*}(\vec{x}) := f(\vec{x}^*) + (\vec{x} - \vec{x}^*)^T \cdot \nabla_{\vec{x}} f(\vec{x}^*) \tag{1.25}$$

We define the second-order Taylor expansion of $f$ around some point $\vec{x}^* \in \mathbb{R}^K$ as the function $\tilde{f}^2_{\vec{x}^*} : \mathbb{R}^K \to \mathbb{R}$ with the following form.

$$\tilde{f}^2_{\vec{x}^*}(\vec{x}) := \tilde{f}^1_{\vec{x}^*}(\vec{x}) + \frac{1}{2}(\vec{x} - \vec{x}^*)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}^*) \cdot (\vec{x} - \vec{x}^*) \tag{1.26}$$

**Theorem 27** (Taylor's theorem (excerpt))**.** *Let $f : \mathbb{R}^K \to \mathbb{R}$ be a twice-differentiable function. For every $\vec{x}^* \in \mathbb{R}^K$ and every $\epsilon > 0$ there exists a $\delta > 0$ such that for all $\vec{x} \in \mathcal{N}_\delta(\vec{x}^*)$ it holds:*

$$|\tilde{f}^2_{\vec{x}^*}(\vec{x}) - f(\vec{x})| < |\tilde{f}^1_{\vec{x}^*}(\vec{x}) - f(\vec{x})| < \epsilon$$

*Proof.* We omit this proof for space constraints. □

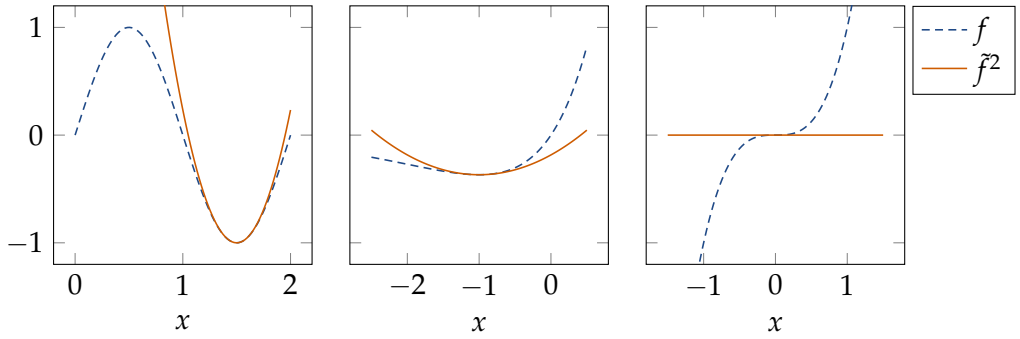



*Figure 1.3:* The second-order Taylor approximation of the functions $f(x) = \sin(\pi \cdot x)$, $f(x) = x \cdot \exp(x)$, and $f(x) = x^3$ at positions 1.5, $-1$ and 0 respectively. The original function is shown as a dashed blue line and the second-order approximation is shown in orange.

**Remark 28** (Intuition behind the Taylor expansion)**.** The previous theorem tells us that the Taylor expansion is a good approximation if we stay in a small $\delta$-neighborhood around our point $\vec{x}^*$ and that the second-order approximation is better than the first-order approximation. Why is that? Intuitively, if a function is continuous, then it changes only a little if we change the input a little, and we can approximate this small change by approximating the function with a tangent. This tangent is described exactly by the first-order Taylor expansion.

The first-order approximation is only inaccurate if the slope changes, i.e. if the first derivative is not constant. However, the first derivative is *also* a continuous function, which means that the slope changes only slightly if we change the input slightly, such that the approximation is accurate if we stay close enough to $\vec{x}^*$.

The second-order approximation is better because it can even approximate constant changes in slope, i.e. a constant second derivative. If the second derivative is not constant, its change will also only change a little if we stay close to to $\vec{x}^*$ and our second-order approximation is good if we stay close enough.

An illustration of the second-order Taylor expansion is also visible in Figure 1.3.

To prove that we can find local minima via the gradient and the Hessian, we only require one final piece of the puzzle, namely the notion of *positive definiteness*, which will tell us that the curvature around a point is positive.

**Definition 29** (Positive (semi-)definiteness)**.** A symmetric square matrix $\boldsymbol{A} \in \mathbb{R}^{K \times K}$ is called *positive definite* if for all vectors $\vec{x} \in \mathbb{R}^K$ with $\vec{x} \neq \vec{0}$ it holds:

$$\vec{x}^T \cdot \boldsymbol{A} \cdot \vec{x} > 0 \tag{1.27}$$

A symmetric square matrix $\boldsymbol{A} \in \mathbb{R}^{K \times K}$ is called *positive semi-definite* of for all vectors $\vec{x} \in \mathbb{R}^K$ it holds:

$$\vec{x}^T \cdot \boldsymbol{A} \cdot \vec{x} \geqslant 0 \tag{1.28}$$

**Remark 30** (Positive definiteness versus positiveness)**.** Importantly, positive definiteness is *not* the same as having positive entries. Consider the matrices

$$\boldsymbol{A}_1 = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{A}_2 = \begin{pmatrix} 5 & -1 \\ -1 & 1.25 \end{pmatrix}.$$

$\boldsymbol{A}_1$ has only positive entries. However, the vector $\vec{x} = (-1, 1)^T$ yields

$$(-1, 1) \cdot \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} -1 \\ 1 \end{pmatrix} = (-1 + 2, -2 + 1) \cdot \begin{pmatrix} -1 \\ 1 \end{pmatrix} = -2 < 0,$$

which means that $\boldsymbol{A}_1$ is *neither* positive semi-definite nor positive definite.

Conversely, $\boldsymbol{A}_2$ is indeed positive definite (and positive semi-definite) even though it contains negative entries. We will show later in Theorem 35 how to check positive definiteness in practice.

### 1.2.2 *Searching for Optima with Gradient and Hessian*

We now go on to show that we can find local minima by searching for locations with zero gradient and positive definite Hessian.

**Theorem 31** (First- and second-order conditions for local minima)**.** *Let $f : \mathbb{R}^K \to \mathbb{R}$ be a twice differentiable function and consider the unconstrained optimization problem in standard form:*

$$\min_{\vec{x} \in \mathbb{R}^K} \quad f(\vec{x})$$

*Then it holds for any $\vec{x}^* \in \mathbb{R}^K$:*

1. *If $\nabla_{\vec{x}} f(\vec{x}^*) \neq 0$, then $\vec{x}^*$ is* not *a local minimum.*
2. *If $\nabla_{\vec{x}} f(\vec{x}^*) = \vec{0}$ and $\nabla^2_{\vec{x}} f(\vec{x}^*)$ is positive definite, then $\vec{x}^*$* is *a local minimum.*
3. *If $\nabla_{\vec{x}} f(\vec{x}^*) = \vec{0}$ and $\nabla^2_{\vec{x}} f(\vec{x}^*)$ is* not *positive semi-definite, then $\vec{x}^*$ is* not *a local minimum.*
4. *If $\vec{x}^*$* is *a local minimum, then $\nabla_{\vec{x}} f(\vec{x}^*) = \vec{0}$ and $\nabla^2_{\vec{x}} f(\vec{x}^*)$ is positive semi-definite.*

*Proof.* Note that we only provide a sketch of the proof here. The precise version would require to ensure that the approximation error of the Taylor expansions stays within certain bounds, which we omit here.

We consider each claim in turn.

1. This follows from the first-order Taylor approximation, i.e. $\tilde{f}^1_{\vec{x}^*}(\vec{x}) = f(\vec{x}^*) + (\vec{x} - \vec{x}^*)^T \cdot \nabla_{\vec{x}} f(\vec{x}^*)$. If the gradient is nonzero, consider the point $\vec{x} = \vec{x}^* - \epsilon \cdot \nabla_{\vec{x}} f(\vec{x}^*)$ for a sufficiently small $\epsilon$ such that the first-order Taylor approximation is accurate. Then it holds

$$f(\vec{x}) = \tilde{f}^1_{\vec{x}^*}(\vec{x}) = f(\vec{x}^*) - \epsilon \cdot \nabla_{\vec{x}} f(\vec{x}^*)^T \cdot \nabla_{\vec{x}} f(\vec{x}^*) < f(\vec{x}^*).$$

   Therefore, for any $\delta > 0$, there is at least one point $\vec{x} \in \mathcal{N}_\delta(\vec{x}^*)$ with $f(\vec{x}) < f(\vec{x}^*)$. This, in turn, implies that $\vec{x}^*$ is not a local minimum.

2. This follows from the second-order Taylor approximation, i.e. $\tilde{f}^2_{\vec{x}^*}(\vec{x}) = f(\vec{x}^*) + \frac{1}{2}(\vec{x} - \vec{x}^*)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}^*) \cdot (\vec{x} - \vec{x}^*)$. Note that the gradient term is removed because the gradient is zero in this case. Now, consider an $\epsilon > 0$ sufficiently small such that this approximation is accurate. Then, because the Hessian is positive definite, it holds for any $\vec{x} \in \mathcal{N}_\epsilon(\vec{x}^*)$ with $\vec{x} \neq \vec{x}^*$:

$$f(\vec{x}) = \tilde{f}^2_{\vec{x}^*}(\vec{x}) = f(\vec{x}^*) + \frac{1}{2}(\vec{x} - \vec{x}^*)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}^*) \cdot (\vec{x} - \vec{x}^*) > f(\vec{x}^*),$$

   such that $\vec{x}^*$ is a local minimum.

3. This follows again from the second-order Taylor approximation. Because the Hessian is *not* positive semi-definite, there must exist a vector $\vec{y} \in \mathbb{R}^K$ such that $\vec{y}^T \cdot \nabla^2_{\vec{x}} f(\vec{x}^*) \cdot \vec{y} < 0$. Then, consider the vector $\vec{x} := \vec{x}^* + \epsilon \cdot \frac{\vec{y}}{\|\vec{y}\|}$. Note that this vector is guaranteed to lie in the $\epsilon$-neighborhood of $\vec{x}^*$ because

$$\|\vec{x} - \vec{x}^*\| = \|\epsilon \cdot \frac{\vec{y}}{\|\vec{y}\|}\| = \frac{\epsilon}{\|\vec{y}\|} \cdot \|\vec{y}\| = \epsilon$$

Further it holds:

$$(\vec{x} - \vec{x}^*)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}^*) \cdot (\vec{x} - \vec{x}^*) = (\epsilon \cdot \frac{\vec{y}}{\|\vec{y}\|})^T \cdot \nabla^2_{\vec{x}} f(\vec{x}^*) \cdot (\epsilon \cdot \frac{\vec{y}}{\|\vec{y}\|}) = \frac{\epsilon^2}{\|\vec{y}\|^2} \cdot \vec{y}^T \cdot \nabla^2_{\vec{x}} f(\vec{x}^*) \cdot \vec{y} < 0$$

Accordingly, for any sufficiently small $\epsilon$ such that the second-order Taylor approximation is accurate it holds: There exists a point $\vec{x} := \vec{x}^* + \epsilon \cdot \frac{\vec{y}}{\|\vec{y}\|}$ in the $\epsilon$-neighborhood of $\vec{x}^*$ such that

$$f(\vec{x}) = \tilde{f}^2_{\vec{x}^*}(\vec{x}) = f(\vec{x}^*) + \frac{1}{2}(\vec{x} - \vec{x}^*)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}^*) \cdot (\vec{x} - \vec{x}^*) < f(\vec{x}^*)$$

which implies that $\vec{x}^*$ is *not* a local minimum.

4. Follows from the negation of 1. and 3.

□

**Remark 32** (Searching for local minima via Theorem 31)**.** Theorem 31 gives us a way to hunt for local minima by setting the gradient to zero and inspecting the Hessian for the resulting solutions – just as we learned in school. It is valuable to inspect the pitfalls of this method, though.

First, we can only apply this method if our optimization problem is *unconstrained*.

Second, it is *not* sufficient that the Hessian is positive semi-definite. We need strict positive definiteness. Otherwise, our point could be either a local minimum, a local maximum, or a saddle point because the function may still be "curved", albeit in higher-order terms which our second-order Taylor expansion can not "see". Also refer to the following examples.

Third, we need to check positive definiteness instead of just positive entries. The effective way of doing so is via the eigenvalues, as we will see in Theorem 35.

**Example 33** (Local minima and Taylor approximation)**.** Consider the examples shown in Figure 1.3.

1. Consider $f(x) = \sin(\pi \cdot x)$. Then, we obtain $\nabla_x f(x) = \cos(\pi \cdot x) \cdot \pi$ and $\nabla^2_x f(x) = -\sin(\pi \cdot x) \cdot \pi^2$. The gradient is zero for any $x^* \in \{0.5 + k \,|\, k \in \mathbb{Z}\}$. First, consider $x^* \in \{0.5 + 2k \,|\, k \in \mathbb{Z}\}$. For these points, we obtain $\nabla^2_x f(x^*) = -\sin(\pi \cdot [0.5 + 2k]) \cdot \pi^2 = -\pi^2 < 0$, i.e. our Hessian is not positive semi-definite and these points are definitely not local minima. Next, consider $x^* \in \{1.5 + 2k \,|\, k \in \mathbb{Z}\}$. For these points, we obtain $\nabla^2_x f(x^*) = -\sin(\pi \cdot [1.5 + 2k]) \cdot \pi^2 = \pi^2 > 0$, i.e. our Hessian is positive definite and these points are definitely local minima (as visible in Figure 1.3, left).

2. Consider $f(x) = x \cdot \exp(x)$. Then, we obtain $\nabla_x f(x) = \exp(x) \cdot (1 + x)$ and $\nabla^2_x f(x) = \exp(x) \cdot (2 + x)$. The gradient is zero for $x^* = -1$. We obtain the Hessian $\nabla^2_x f(x^*) = \exp(-1) \cdot (2 - 1) = \exp(-1) > 0$, i.e. our Hessian is positive definite and this point is definitely a local minimum (as visible in Figure 1.3, center).

3. Consider $f(x) = x^3$. Then, we obtain $\nabla_x f(x) = 3 \cdot x^2$ and $\nabla_x^2 f(x) = 6 \cdot x$. The gradient is zero for $x^* = 0$. We obtain the Hessian $\nabla_x^2 f(x^*) = 0$, i.e. our Hessian is positive semi-definite. Still, this point is clearly not a local minimum (as visible in Figure 1.3, right).

In addition, consider the following examples.

1. Consider $f(x) = x^4$. Then, we obtain $\nabla_x f(x) = 4 \cdot x^3$ and $\nabla_x^2 f(x) = 12 \cdot x^2$. The gradient is zero for $x^* = 0$. We obtain the Hessian $\nabla_x^2 f(x^*) = 0$, i.e. our Hessian is positive semi-definite. Still, this point is a global minimum because for any $x \in \mathbb{R}$ we have $f(x) = x^4 \geqslant 0 = f(\vec{x}^*)$.

2. Consider $f(x) = -x^4$. Then, we obtain $\nabla_x f(x) = -4 \cdot x^3$ and $\nabla_x^2 f(x) = -12 \cdot x^2$. The gradient is zero for $x^* = 0$. We obtain the Hessian $\nabla_x^2 f(x^*) = 0$, i.e. our Hessian is positive semi-definite. However, out point is indeed a global maximum because for any $x \in \mathbb{R}$ we have $f(x) = -x^4 \leqslant 0 = f(x^*)$.

Finally, consider the two-dimensional function $f(x, y) = x^2 - y^2$. In this case, we obtain

$$\nabla_{(x,y)} f(x, y) = \begin{pmatrix} 2x \\ -2y \end{pmatrix} \quad \text{and} \quad \nabla_{(x,y)}^2 f(x, y) = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}.$$

The gradient is zero for $\vec{x}^* = (0, 0)^T$. The Hessian is *not* positive semi-definite because we find that for $\vec{y} = (0, 1)^T$ we obtain

$$(0, 1) \cdot \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = (0, -2) \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = -2 < 0$$

Therefore, $\vec{x}^*$ is *not* a local minimum.

### 1.2.3 *Eigenvalue analysis*

Until now we are missing an efficient way to check whether a matrix is positive definite or positive semi-definite. As it turns out, the key to such a method are *eigenvalues*.

**Remark 34** (Refresher: Eigenvalues)**.** As a refresher: An eigen*vector* of a matrix $\boldsymbol{A}$ is defined as a vector $\vec{v} \neq \vec{0}$, such that

$$\boldsymbol{A} \cdot \vec{v} = \lambda \cdot \vec{v} \tag{1.29}$$

where $\lambda \in \mathbb{R}$ is called the *eigenvalue* corresponding to the eigenvector $\vec{v}$. Note that if $\vec{v}$ is an eigenvalue of $\boldsymbol{A}$, then $\alpha \cdot \vec{v}$ for any $\alpha \neq 0$ is also an eigenvector of $\boldsymbol{A}$ with the same eigenvalue.

Further, we can re-write any symmetric square matrix $\boldsymbol{A} \in \mathbb{R}^{K \times K}$ as follows.

$$\boldsymbol{A} = \boldsymbol{V} \cdot \boldsymbol{\Lambda} \cdot \boldsymbol{V}^T \tag{1.30}$$

where $\boldsymbol{V}$ contains eigenvectors of $\boldsymbol{A}$ as columns and $\Lambda$ is a diagonal matrix containing the corresponding eigenvalues. This form is also called the *eigendecomposition* of $\boldsymbol{A}$ and can be computed in $\mathcal{O}(K^3)$. A special property of this eigenvalue decomposition is that the eigenvectors are orthogonal, that is: $\boldsymbol{V} \cdot \boldsymbol{V}^T = \boldsymbol{V}^T \cdot \boldsymbol{V} = \boldsymbol{I}^K$, where $\boldsymbol{I}^K$ is the $K \times K$-dimensional identity matrix.

With this knowledge in mind, we can now go on to prove that knowing the eigenvalues of $\boldsymbol{A}$ is sufficient to know its definiteness.

**Theorem 35** (Positive definiteness and eigenvalues)**.** *A symmetric square matrix $\boldsymbol{A} \in \mathbb{R}^{K \times K}$ is positive definite if and only if all of its eigenvalues are positive.*

*A symmetric square matrix $\boldsymbol{A} \in \mathbb{R}^{K \times K}$ is positive semi-definite if and only if all of its eigenvalues are non-negative.*

*Proof.* Since $\boldsymbol{A}$ is symmetric and square, the eigenvalue decomposition of $\boldsymbol{A}$ has the form $\boldsymbol{A} = \boldsymbol{V} \cdot \boldsymbol{\Lambda} \cdot \boldsymbol{V}^T$ with $\boldsymbol{V} \cdot \boldsymbol{V}^T = \boldsymbol{V}^T \cdot \boldsymbol{V} = \boldsymbol{I}^K$.

We first prove the second claim.

If all eigenvalues are non-negative, we can re-write $\boldsymbol{\Lambda}$ as $\sqrt{\boldsymbol{\Lambda}}^T \cdot \sqrt{\boldsymbol{\Lambda}}$ where $\sqrt{\cdot}$ denotes the element-wise square root. Accordingly, we can re-write for all $\vec{x} \in \mathbb{R}^K$:

$$\vec{x}^T \cdot \boldsymbol{A} \cdot \vec{x} = \vec{x}^T \cdot \boldsymbol{V} \cdot \sqrt{\boldsymbol{\Lambda}}^T \cdot \sqrt{\boldsymbol{\Lambda}} \cdot \boldsymbol{V}^T \cdot \vec{x} = \left(\sqrt{\boldsymbol{\Lambda}} \cdot \boldsymbol{V}^T \cdot \vec{x}\right)^T \cdot \left(\sqrt{\boldsymbol{\Lambda}} \cdot \boldsymbol{V}^T \cdot \vec{x}\right) \geqslant 0,$$

which means that $\boldsymbol{A}$ is positive semi-definite.

Conversely, if there exists a negative eigenvalue of $\boldsymbol{A}$, say $\lambda_k < 0$, then we can consider the $k$th eigenvector $\vec{v}_k$ and the product

$$\vec{v}_k^T \cdot \boldsymbol{A} \cdot \vec{v}_k = \vec{v}_k^T \cdot \lambda_k \cdot \vec{v}_k = \lambda_k < 0$$

which means that $\boldsymbol{A}$ is *not* positive semi-definite.

Now, consider the first claim. If all eigenvalues are positive, then $\sqrt{\boldsymbol{\Lambda}} \cdot \boldsymbol{V}^T \cdot \vec{x}$ is nonzero if $\vec{x}$ is nonzero. Therefore, for any nonzero $\vec{x}$ we obtain

$$\vec{x}^T \cdot \boldsymbol{A} \cdot \vec{x} = \left(\sqrt{\boldsymbol{\Lambda}} \cdot \boldsymbol{V}^T \cdot \vec{x}\right)^T \cdot \left(\sqrt{\boldsymbol{\Lambda}} \cdot \boldsymbol{V}^T \cdot \vec{x}\right) > 0,$$

which means that $\boldsymbol{A}$ is positive definite.

Conversely, consider the case of a non-positive eigenvalue $\lambda_k$. If $\lambda_k < 0$, then $\boldsymbol{A}$ is not even positive semi-definite, which means that it is also not positive definite (see above). if $\lambda_k = 0$ then consider the product

$$\vec{v}_k^T \cdot \boldsymbol{A} \cdot \vec{v}_k = \vec{v}_k^T \cdot \lambda_k \cdot \vec{v}_k = 0$$

which means that $\boldsymbol{A}$ is *not* positive definite. □

**Remark 36** (Eigenvalues of the Hessian and curvature)**.** Intuitively, the eigenvalues of the Hessian specify the curvature of the function along the direction of the corresponding eigenvectors. Positive eigenvalues correspond to upwards curvature, negative eigenvalues to downwareds curvature.

If the curvature along all directions is upwards - that is, if all eigenvalues of the Hessian are positive - we can only increase the objective function if we move. However, if even one eigenvalue is negative (or zero), we can move into that direction and decrease the objective function (refer to Figure 1.4).

Note that this interpretation *only* applies for points where the gradient is zero. In other points, the eigenvalues do *not* correspond to curvature. Also note that this interpretation only holds for *quadratic* curvature. Higher-order curvatures are not included.

In this section, we have demonstrated how we can find local minima in continuous optimization. This begs the question: Under which circumstances can we guarantee that local minima are also global minima? In this regard, a subclass of continuous optimization problems comes into play, namely the class of *convex* optimization problems.

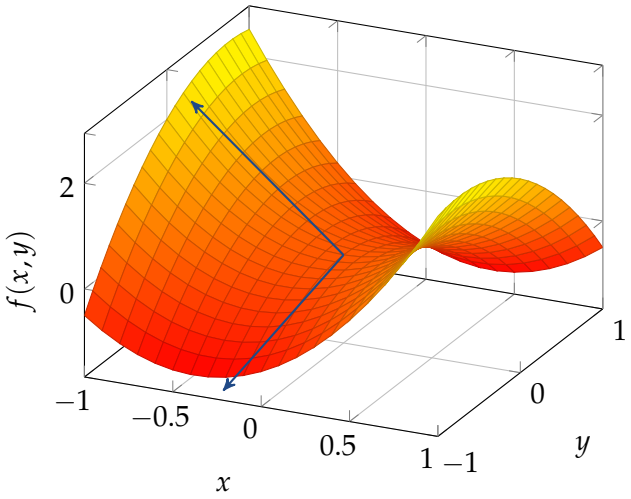



*Figure 1.4:* An illustration of curvature at a point with zero gradient. The function here is $f(x,y) = 2x^2 - \frac{3}{2}x \cdot y - y^2$, indicated by the surface plot. The arrows indicate the eigenvectors of the Hessian at point $(0,0)$, where the $z$ coordinate indicates the (square root of) the corresponding eigenvalue. Note that the eigenvalues correspond to the curvature around this point, being positive for upward and negative for downward curvature.

## 1.3 CONVEX OPTIMIZATION

Convex Optimization is concerned with problems that adhere to certain geometric constraints (namely convexity). Why do we care about such geometric constraints? Because they guarantee a very useful property for optimization, namely that all local minima are global minima. Since finding local minima is usually easy in continuous optimization (see previous section), having this property makes optimization easy. Accordingly, it is very desirable to re-phrase optimization problems in convex form.

### 1.3.1 *Definition and Convex Optimization Theorem*

In this section, we will define precisely what convexity means and then prove that local minima are global minima for convex problems.

**Definition 37** (Convex set, convex function, convex optimization problem)**.** Let $\mathcal{X} \subseteq \mathbb{R}^K$ for some $K \in \mathbb{N}$. We call $\mathcal{X}$ a *convex set* if for all $\vec{x}, \vec{y} \in \mathcal{X}$ and all $\alpha \in [0,1]$ it holds:

$$\alpha \cdot \vec{y} + (1-\alpha) \cdot \vec{x} \in \mathcal{X}. \tag{1.31}$$

Let $f$ be a function $f : \mathcal{X} \to \mathbb{R}$. We call $f$ a *convex function* if $\mathcal{X}$ is convex and for all $\vec{x}, \vec{y} \in \mathcal{X}$ and all $\alpha \in [0,1]$ it holds:

$$f\big(\alpha \cdot \vec{y} + (1-\alpha) \cdot \vec{x}\big) \leqslant \alpha \cdot f(\vec{y}) + (1-\alpha) \cdot f(\vec{x}). \tag{1.32}$$

Finally, consider a optimization problem in standard form. We call this problem *convex* if its objective function is a convex function and its feasible set is a convex set.

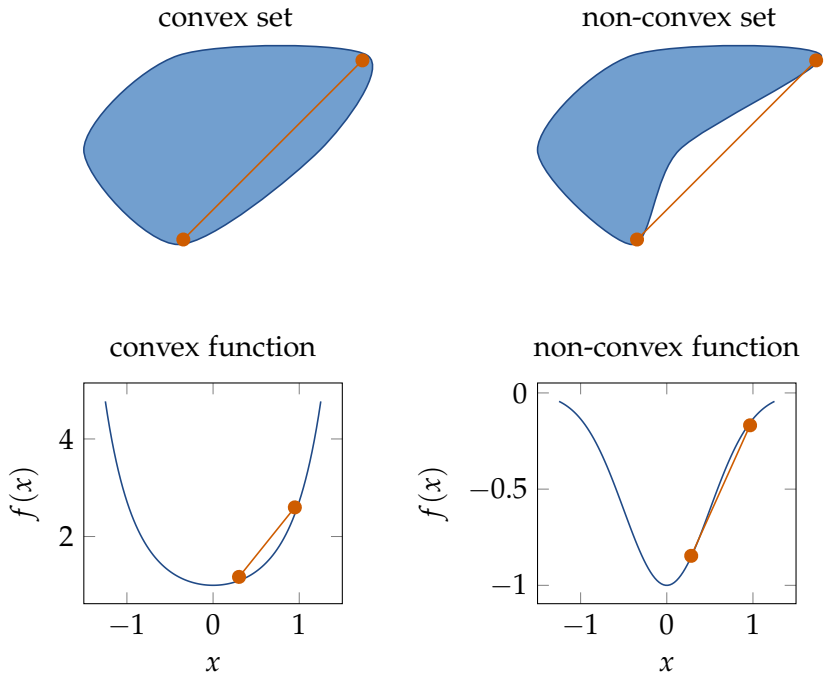



*Figure 1.5:* Top left: A convex set where every connecting line lies in the set; top right: a non-convex set due to two points for which the connecting line does *not* lie in the set; bottom left: A convex function where every connecting line lies above the graph; bottom right: A non-convex function where for at least two points the connecting line is *not* above the graph.

**Remark 38** (Geometric intuition of convexity)**.** In plain English, a set is convex if the connecting line between any two points in the set is also in the set. The points on the connecting line are precisely described by Equation 1.31. Also refer to Figure 1.5 (top).

Similarly, a function is convex if the connecting line between two points on its graph is above (or at least not below) the graph itself. This condition is made precise by Equation 1.32. Also refer to Figure 1.5 (bottom).

**Example 39** (Convex sets, convex functions)**.** The following sets are known to be convex

- the $K$-dimensional vector space $\mathbb{R}^K$ for any $K \in \mathbb{N}$,
- all triangles and all rectangles,
- all $n$-dimensional balls, and
- all platonic solids.

The following multivariate functions are known to be convex.

- Linear (actually affine) functions, i.e. $f(\vec{x}) = \vec{c}^T \cdot \vec{x} + b$ for some vector $\vec{c}$ and some scalar $b$, and
- Positive semi-definite quadratic functions, i.e. $f(\vec{x}) = \frac{1}{2}\vec{x}^T \cdot \boldsymbol{P} \cdot \vec{x} + \vec{q}^T \cdot \vec{x} + r$ for some positive semi-definite matrix $\boldsymbol{P}$, some vector $\vec{q}$ and some scalar $r$

The following univariate functions are known to be convex.

- Linear (actually affine) functions, i.e. $f(x) = c \cdot x + b$ for some $c, b \in \mathbb{R}$,
- Polynomials of even degree, i.e. $f(x) = x^{2 \cdot k}$ for every $k \in \mathbb{N}$,
- The exponential function $f(x) = \exp(x)$,
- The negative logarithm $f(x) = -\log(x)$, and
- The absolute value function $f(x) = |x|$.

If we know that an optimization problem is convex, we can show that optimization is simple.

**Theorem 40** (Convex optimization theorem)**.** *For a convex optimization problem, all local minima are global minima.*

*Proof.* Assume the claim is *not* true. Then, there exists a $\vec{x}^* \in \mathcal{X}^*$ which is a local minimum but *not* a global minimum. Accordingly, there exists at least one $\vec{y} \in \mathcal{X}^*$ for which $f(\vec{y}) < f(\vec{x}^*)$. Now, consider the point $\vec{x}_\alpha := \alpha \cdot \vec{y} + (1 - \alpha) \cdot \vec{x}^*$ which lies on the connecting line between $\vec{x}^*$ and $\vec{y}$. Because $\mathcal{X}^*$ is convex, $\vec{x}_\alpha \in \mathcal{X}^*$ for all $\alpha \in [0, 1]$. Now, consider the distance between $\vec{x}_\alpha$ and $\vec{x}^*$, which is

$$\|\vec{x}_\alpha - \vec{x}^*\| = \|\alpha \cdot \vec{y} - \alpha \cdot \vec{x}^*\| = \alpha \cdot \|\vec{y} - \vec{x}^*\|$$

Note that $\|\vec{y} - \vec{x}^*\|$ is a constant. Thus, by reducing $\alpha$, we can bring $\vec{x}_\alpha$ arbitrarily close to $\vec{x}^*$. In particular, by setting $\alpha = \epsilon / \|\vec{y} - \vec{x}^*\|$, we can ensure that $\vec{x}_\alpha \in \mathcal{N}_\epsilon(\vec{x}^*)$ for any $\epsilon > 0$. Because $\vec{x}^*$ is a local minimum, there exists an $\epsilon$ such that for all $\vec{x} \in \mathcal{N}_\epsilon(\vec{x}^*) \cap \mathcal{X}^*$ it holds $f(\vec{x}^*) \leqslant f(\vec{x})$. Therefore, for $\alpha = \epsilon / \|\vec{y} - \vec{x}^*\|$, we obtain $f(\vec{x}^*) \leqslant f(\vec{x}_\alpha)$. However, because $f$ is a convex function, we also know that

$$f(\vec{x}^*) \leqslant f(\vec{x}_\alpha) \leqslant \alpha \cdot f(\vec{y}) + (1 - \alpha) \cdot f(\vec{x}^*) < \alpha \cdot f(\vec{x}^*) + (1 - \alpha) \cdot f(\vec{x}^*) = f(\vec{x}^*),$$

which is a contradiction. Therefore, $\vec{x}^*$ must be a global minimum. □

### 1.3.2 *Engineering Convex Problems*

Now that we know that convexity is useful, the immediate question is how we can make an optimization problem convex. In a first step, we will provide tools by which we can show that a *function* is convex. After that, we will show how to construct a convex optimization problem from convex functions.

**Theorem 41** (First- and second-order conditions of convex functions)**.** *Let $\mathcal{X} \subseteq \mathbb{R}^K$ be a convex set and let $f$ be a function $f : \mathcal{X} \to \mathbb{R}$. Then it holds:*

1. *$f$ is convex if and only if for all $\vec{x}, \vec{y} \in \mathcal{X}$ with $\vec{x} \neq \vec{y}$ it holds:*

$$f(\vec{y}) > \tilde{f}^1_{\vec{x}}(\vec{y}) = f(\vec{x}) + (\vec{y} - \vec{x})^T \cdot \nabla_{\vec{x}} f(\vec{x}). \tag{1.33}$$

2. *$f$ is convex if and only if for all $\vec{x} \in \mathcal{X}$ it holds: $\nabla^2_{\vec{x}} f(\vec{x})$ is positive semi-definite.*

*Proof.* Our proof of the first claim is inspired by Boyd and Vandenberghe (2004, p. 70).

We first introduce two auxiliary concepts. Let $\vec{x}, \vec{y} \in \mathcal{X}$ and $\alpha \in [0,1]$. Then, we define $\vec{z}_\alpha := \alpha \cdot \vec{y} + (1-\alpha) \cdot \vec{x}$. Further, we define the function $\ell : [0,1] \to \mathbb{R}$ as the one-dimensional line along the function $f$ which connects $f(\vec{x})$ and $f(\vec{y})$, i.e. $\ell(\alpha) := f(\vec{z}_\alpha)$. Note that we obtain for the derivative

$$\frac{\partial}{\partial \alpha} \ell(\alpha) = \left(\nabla_{\vec{z}_\alpha} f(\vec{z}_\alpha)\right)^T \cdot \frac{\partial}{\partial \alpha} \vec{z}_\alpha = \left(\nabla_{\vec{z}_\alpha} f(\vec{z}_\alpha)\right)^T \cdot (\vec{y} - \vec{x})$$

Assume first that $f$ is convex, i.e. Equation 1.32 holds. Then, $\ell$ is convex as well. Consider now $\alpha, \beta, \Delta \in [0,1]$. Then, because $\ell$ is convex, we obtain:

$$\begin{aligned} & \ell\big(\Delta \cdot \beta + (1-\Delta) \cdot \alpha\big) \leqslant \Delta \cdot \ell(\beta) + (1-\Delta) \cdot \ell(\alpha) \\ \iff & \frac{\ell\big(\Delta \cdot \beta + \alpha - \Delta \cdot \alpha\big)}{\Delta} \leqslant \ell(\beta) + \frac{\ell(\alpha)}{\Delta} - \ell(\alpha) \\ \iff & \ell(\alpha) + \frac{\ell\big(\alpha + \Delta \cdot (\beta - \alpha)\big) - \ell(\alpha)}{\Delta} \leqslant \ell(\beta) \end{aligned}$$

Since this inequality holds for all $\Delta \in [0,1]$, we can consider the limes towards zero, i.e.:

$$\ell(\beta) \geqslant \ell(\alpha) + \lim_{\Delta \to 0} \frac{\ell\big(\alpha + \Delta \cdot (\beta - \alpha)\big) - \ell(\alpha)}{\Delta} = \ell(\alpha) + \frac{\partial}{\partial \alpha} \ell(\alpha) \cdot (\beta - \alpha)$$

where the last equality holds due to the definition of a derivative and the chain rule. Since this result holds for any two $\alpha, \beta \in [0,1]$, it also holds for the special case $(\beta, \alpha) = (1, 0)$, i.e.:

$$\begin{aligned} & \ell(1) \geqslant \ell(0) + \frac{\partial}{\partial \alpha} \ell(0) \cdot (1-0) \\ \iff & f(\vec{z}_1) \geqslant f(\vec{z}_0) + \left(\nabla_{\vec{z}_0} f(\vec{z}_0)\right)^T \cdot (\vec{y} - \vec{x}) \\ \iff & f(\vec{y}) \geqslant f(\vec{x}) + \left(\nabla_{\vec{x}} f(\vec{x})\right)^T \cdot (\vec{y} - \vec{x}), \end{aligned}$$

which is exactly Inequality 1.33.

Now, assume that Inequality 1.33 holds. Because $\mathcal{X}$ is convex, $\vec{z}_\alpha, \vec{z}_\beta \in \mathcal{X}$ for any two $\alpha, \beta \in [0,1]$. Further, we obtain:

$$\vec{z}_\beta - \vec{z}_\alpha = \beta \cdot \vec{y} + (1-\beta) \cdot \vec{x} - \alpha \cdot \vec{y} - (1-\alpha) \cdot \vec{x} = \beta \cdot (\vec{y} - \vec{x}) + \vec{x} - \alpha \cdot (\vec{y} - \vec{x}) - \vec{x} = (\vec{y} - \vec{x}) \cdot (\beta - \alpha)$$

From Inequality 1.33 we obtain:

$$\begin{aligned} & f(\vec{z}_\beta) \geqslant f(\vec{z}_\alpha) + \left(\nabla_{\vec{x}} f(\vec{z}_\alpha)\right)^T \cdot (\vec{z}_\beta - \vec{z}_\alpha) \\ \iff & \ell(\beta) \geqslant \ell(\alpha) + \left(\nabla_{\vec{x}} f(\vec{z}_\alpha)\right)^T \cdot (\vec{y} - \vec{x}) \cdot (\beta - \alpha) \\ \iff & \ell(\beta) \geqslant \ell(\alpha) + \frac{\partial}{\partial \alpha} \ell(\alpha) \cdot (\beta - \alpha) \end{aligned} \tag{1.34}$$

Now, let $\Delta \in [0,1]$ and consider two instances of Inequality 1.34, first with $(\beta, \alpha) = (1, \Delta)$, and second with $(\beta, \alpha) = (0, \Delta)$. This yields:

$$\ell(1) \geqslant \ell(\Delta) + \frac{\partial}{\partial \Delta} \ell(\Delta) \cdot (1 - \Delta) \qquad \text{and}$$
$$\ell(0) \geqslant \ell(\Delta) + \frac{\partial}{\partial \Delta} \ell(\Delta) \cdot (0 - \Delta)$$

If we multiply the first inequality with $\Delta$, the second inequality with $(1-\Delta)$ and add both, we obtain:

$$\begin{aligned}
&\Delta \cdot \ell(1) + (1-\Delta)\cdot \ell(0) \geqslant \Delta \cdot \ell(\Delta) + \frac{\partial}{\partial \Delta}\ell(\Delta)\cdot(1-\Delta)\cdot\Delta \\
&\qquad\qquad + (1-\Delta)\cdot\ell(\Delta) - \frac{\partial}{\partial\Delta}\ell(\Delta)\cdot\Delta\cdot(1-\Delta) \\
\iff\ & \Delta\cdot f(\vec{z}_1) + (1-\Delta)\cdot f(\vec{z}_0) \geqslant f(\vec{z}_\Delta) \\
\iff\ & \Delta\cdot f(\vec{y}) + (1-\Delta)\cdot f(\vec{x}) \geqslant f\big(\Delta\cdot\vec{y} + (1-\Delta)\cdot\vec{x}\big),
\end{aligned}$$

which is exactly Inequality 1.32.

Now, consider the second claim. First, assume that there exists some $\vec{x} \in \mathcal{X}$ such that $\nabla^2_{\vec{x}} f(\vec{x})$ is *not* positive semi-definite. Then, there exists some $\vec{y} \in \mathbb{R}^K$ such that $\vec{y}^T \cdot \nabla^2_{\vec{x}} f(\vec{x}) \cdot \vec{y} < 0$. Now, consider $\vec{z}_\alpha := \alpha\cdot\vec{y} + \vec{x}$. For sufficiently small $\alpha \in (0,1]$, $\vec{z}_\alpha \in \mathcal{X}$ and the second-order Taylor approximation is precise, i.e.:

$$\begin{aligned}
f(\vec{z}_\alpha) &= f(\vec{x}) + \nabla_{\vec{x}} f(\vec{x})^T \cdot (\vec{z}_\alpha - \vec{x}) + \frac{1}{2}(\vec{z}_\alpha - \vec{x})^T \cdot \nabla^2_{\vec{x}} f(\vec{x}) \cdot (\vec{z}_\alpha - \vec{x}) \\
&= f(\vec{x}) + \nabla_{\vec{x}} f(\vec{x})^T \cdot (\vec{z}_\alpha - \vec{x}) + \frac{1}{2}\alpha^2 \vec{y}^T \cdot \nabla^2_{\vec{x}} f(\vec{x}) \cdot \vec{y} \\
&< f(\vec{x}) + \nabla_{\vec{x}} f(\vec{x})^T \cdot (\vec{z}_\alpha - \vec{x})
\end{aligned}$$

Now, if $f$ were convex, we would obtain:

$$f(\vec{x}) + \nabla_{\vec{x}} f(\vec{x})^T \cdot (\vec{z}_\alpha - \vec{x}) \leqslant f(\vec{z}_\alpha)$$

due to the first claim. However, this is a contradiction because then $f(\vec{z}_\alpha) < f(\vec{z}_\alpha)$. Therefore, $f$ cannot be convex.

Finally, consider the case that for all $\vec{x} \in \mathcal{X}$ $\nabla^2_{\vec{x}} f(\vec{x})$ *is* positive semi-definite. Note that for any two $\vec{x}, \vec{y} \in \mathcal{X}$ there exists some $\alpha \in [0,1]$, such that the second-order Taylor approximation is precise for the Hessian taken at $\alpha\cdot\vec{y} + (1-\alpha)\cdot\vec{x}$, i.e.:

$$f(\vec{y}) = f(\vec{x}) + \nabla_{\vec{x}} f(\vec{x})^T \cdot (\vec{y} - \vec{x}) + \frac{1}{2}(\vec{y} - \vec{x})^T \cdot \nabla^2_{\vec{x}} f(\alpha\cdot\vec{y} + (1-\alpha)\cdot\vec{x}) \cdot (\vec{y} - \vec{x})$$

Because $\nabla^2_{\vec{x}} f(\vec{x})$ is positive semi-definite, the quadratic term is guaranteed to be non-negative. Therefore, we obtain:

$$f(\vec{y}) \geqslant f(\vec{x}) + \nabla_{\vec{x}} f(\vec{x})^T \cdot (\vec{y} - \vec{x}),$$

which implies convexity by virtue of the first claim. □

Now that we know how to prove that a function is convex, we next show how to construct convex optimization problems from convex functions.

**Theorem 42** (Convex constraints yield a convex set)**.** *Consider a continuous optimization problem with inequality constraint functions $g_1, \ldots, g_m$ and equality constraint functions $h_1, \ldots, h_n$. Then, if the two following conditions hold, the feasible set of this problem is convex.*

1. *For all $i \in \{1, \ldots, m\}$, $-g_i$ is convex and*
2. *for all $j \in \{1, \ldots, n\}$, $h_j$ is affine, i.e. there exists some vector $\vec{a}_j$ and some scalar $b_j$ such that $h_j(\vec{x}) = \vec{a}_j^T \cdot \vec{x} + b_j$.*

*Proof.* We will first consider the $i$th inequality constraint. In particular, consider the set

$$\mathcal{F}_i := \{\vec{x} \in \mathbb{R}^K | g_i(\vec{x}) \geqslant 0\} = \{\vec{x} \in \mathbb{R}^K | -g_i(\vec{x}) \leqslant 0\}$$

For any two $\vec{x}, \vec{y} \in \mathcal{F}_i$ it holds: $-g_i(\vec{x}) \leqslant 0$ and $-g_i(\vec{y}) \leqslant 0$. Accordingly, for any $\alpha \in [0,1]$ it also holds:

$$-\alpha \cdot g_i(\vec{y}) - (1-\alpha) \cdot g_i(\vec{x}) \leqslant 0$$

Further, because $-g_i$ is convex, we obtain:

$$0 \geqslant -\alpha \cdot g_i(\vec{y}) - (1-\alpha) \cdot g_i(\vec{x}) \geqslant -g_i\big(\alpha \cdot \vec{y} + (1-\alpha) \cdot \vec{x}\big),$$

which in turn implies that $\alpha \cdot \vec{y} + (1-\alpha) \cdot \vec{x} \in \mathcal{F}_i$. Therefore, $\mathcal{F}_i$ is a convex set for every $i$.

Next, note that the intersection of any two convex sets $\mathcal{A}$ and $\mathcal{B}$ is also convex. Consider two points $\vec{x}, \vec{y} \in \mathcal{A} \cap \mathcal{B}$. Then both $\vec{x}$ and $\vec{y}$ lie in both $\mathcal{A}$ and $\mathcal{B}$ due to the definition of an intersection. Further it holds: For any $\alpha \in [0,1]$ $\alpha \cdot \vec{x} + (1-\alpha) \in \vec{y}$ lies both in $\mathcal{A}$ and in $\mathcal{B}$ because both $\mathcal{A}$ and $\mathcal{B}$ are convex. Therefore, $\alpha \cdot \vec{x} + (1-\alpha) \in \vec{y} \in \mathcal{A} \cap \mathcal{B}$.

Now, consider the $j$th equality constraint. In particular, consider the set

$$\mathcal{G}_j := \{\vec{x} \in \mathbb{R}^K | h_j(\vec{x}) = 0\} = \{\vec{x} \in \mathbb{R}^K | h_j(\vec{x}) \leqslant 0\} \cap \{\vec{x} \in \mathbb{R}^K | -h_j(\vec{x}) \leqslant 0\}$$

Because $h_j$ is affine, both $h_j$ and $-h_j$ are convex and therefore the left and right set in the equation above are both convex by the same reasoning as above. Further, because the intersection of two convex sets is convex, $\mathcal{G}_j$ is convex for every $j$.

Now, note that the entire feasible set can be re-written as:

$$\mathcal{X}^* = \Big(\bigcap_{i=1}^{m} \mathcal{F}_i\Big) \cap \Big(\bigcap_{j=1}^{n} \mathcal{G}_j\Big)$$

Since the intersection of convex sets is convex, $\mathcal{X}^*$ is convex. □

Sometimes, we may be unable to construct our problem from convex functions right away. In these cases, however, it may still be possible to *transform* our objective function, inequality constraint functions, and equality constraint functions to become convex.

**Theorem 43** (Transformer theorem)**.** *Let*

$$\begin{aligned} \min_{\vec{x} \in \mathbb{R}^K} \quad & f(\vec{x}) \\ s.t. \quad & g_i(\vec{x}) \geqslant 0 && \forall i \in \{1, \dots, m\} \\ & h_j(\vec{x}) = 0 && \forall j \in \{1, \dots, n\} \end{aligned}$$

*be an optimization problem in standard form. Further, let $\phi, \rho_1, \dots, \rho_m$ and $\psi_1, \dots, \psi_n$ be strictly montonously increasing functions from $\mathbb{R}$ to $\mathbb{R}$. Then, the following optimization problems is equivalent (in the sense of Definition 7).*

$$\begin{aligned} \min_{\vec{x} \in \mathbb{R}^K} \quad & \phi\big(f(\vec{x})\big) && (1.35) \\ s.t. \quad & \rho_i\big(g_i(\vec{x})\big) - \rho_i(0) \geqslant 0 && \forall i \in \{1, \dots, m\} \\ & \psi_j\big(h_j(\vec{x})\big) - \psi_j(0) = 0 && \forall j \in \{1, \dots, n\} \end{aligned}$$

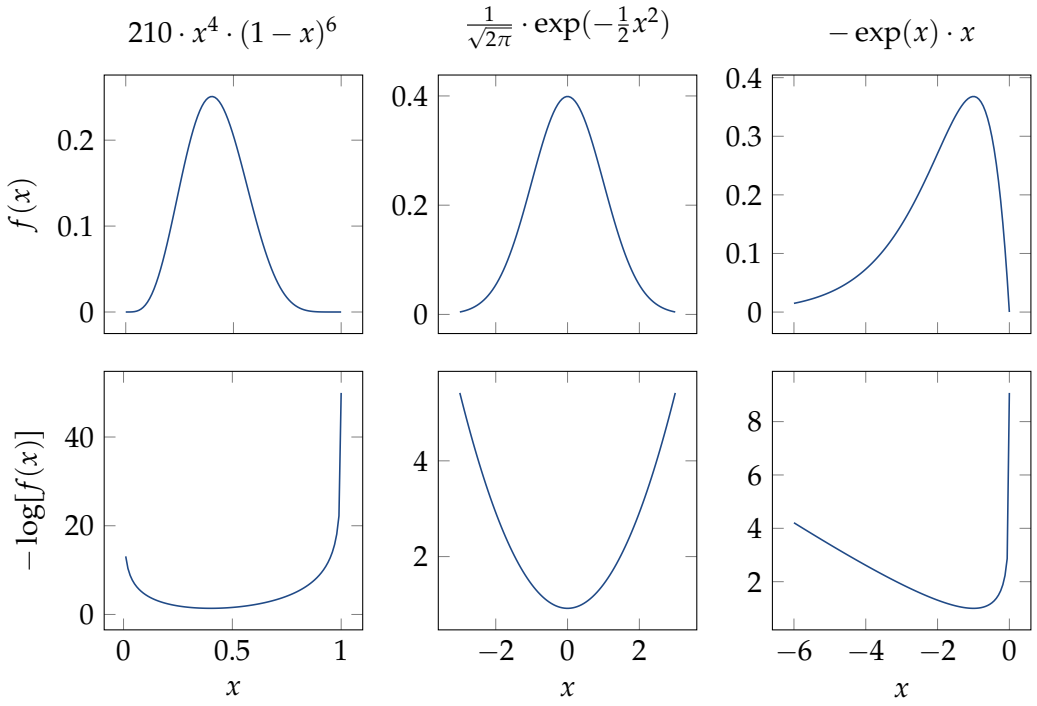



*Figure 1.6:* Three non-convex functions (top) with convex negative logarithms (bottom).

*Proof.* To show equivalence, we need to show that Equations 1.9, 1.10, and 1.11 hold. Equation 1.9 holds trivially because $\phi$ is strictly monotonously increasing.

Regarding Equation 1.10 note that strictly monotonously increasing functions are injective and thus invertible (if restricted to their image). Therefore, we obtain:

$$g_i(\vec{x}) \geqslant 0 \iff \rho_i\big(g_i(\vec{x})\big) \geqslant \rho_i(0) \iff \rho_i\big(g_i(\vec{x})\big) - \rho_i(0) \geqslant 0$$

Using the same invertibility reasoning, we also know for all $j$ that

$$h_j(\vec{x}) = 0 \iff \psi_j\big(h_j(\vec{x})\big) = \psi_j(0) \iff \psi_j\big(h_j(\vec{x})\big) - \psi_j(0) = 0$$

□

**Example 44** (Log-convex functions)**.** The following functions are not convex but become convex by applying a negative logarithm:

- binomial density functions: $f(x) = \binom{n}{k} x^k \cdot (1-x)^{n-k}$ for $x \in [0,1]$, with the negative logarithm $-\log[f(x)] = -\log[\binom{n}{k}] - k \cdot \log[x] - (n-k) \cdot \log[1-x]$.
- Gaussian density functions: $f(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x^2)$, with the negative logarithm $-\log[f(x)] = \frac{1}{2}\log[2\pi] + \frac{1}{2}x^2$.
- $f(x) = -x \cdot \exp(x)$ with the negative logarithm $-\log[f(x)] = -\log[-x] - x$.

Also refer to Figure 1.6.

## 1.4 DUALITY

While differentiable optimization gives us a tool to solve unconstrained, continuous optimization problems, we are still missing an equivalent tool for *constrained* optimization problems. The main trick to address such problems is to re-write them as unconstrained problems or at least less severely constrained problems. We call such a re-written form of an optimization problem its *dual form*. In this section, we will cover two kinds of dual forms, namely the Lagrange and the Wolfe dual, and we will also establish the equivalence of primal and dual form under certain conditions.

### 1.4.1 *Lagrange Dual Form*

**Definition 45** (Lagrange dual)**.** Let

$$\begin{aligned} \min_{\vec{x}\in\mathbb{R}^K} \quad & f(\vec{x}) \\ \text{s.t.} \quad & g_i(\vec{x}) \geqslant 0 && \forall i \in \{1,\dots,m\} \\ & h_j(\vec{x}) = 0 && \forall j \in \{1,\dots,n\} \end{aligned}$$

be an optimization problem in standard form. Then, we define the Lagrange dual of the problem as follows.

$$\begin{aligned} \sup_{\vec{\lambda}\in\mathbb{R}^m,\vec{\mu}\in\mathbb{R}^n} \quad \inf_{\vec{x}\in\mathbb{R}^K} \quad & f(\vec{x}) - \sum_{i=1}^{m} \lambda_i \cdot g_i(\vec{x}) - \sum_{j=1}^{n} \mu_j \cdot h_j(\vec{x}) && (1.36) \\ \text{s.t.} \quad & \lambda_i \geqslant 0 && \forall i \in \{1,\dots,m\} \end{aligned}$$

Whenever we consider a Lagrange dual, we call the original problem the *primal* problem.

We call the function

$$\begin{aligned} \mathcal{L} :& \mathbb{R}^K \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R} && \text{where} \\ \mathcal{L}(\vec{x},\vec{\lambda},\vec{\mu}) :=& f(\vec{x}) - \sum_{i=1}^{m} \lambda_i \cdot g_i(\vec{x}) - \sum_{j=1}^{n} \mu_j \cdot h_j(\vec{x}) && (1.37) \end{aligned}$$

the *Lagrangian* of the problem.

We call the variables $\lambda_i$ and $\mu_j$ the *Lagrange multipliers* of the Lagrange dual.

**Remark 46** (Infimum and supremum)**.** We introduced supremum and infimum notation here, which is necessary for mathematical reasons. Broadly speaking, however, sup is the same as max and inf the same as min, just that sup and inf can also obtain infinity values.

**Example 47** (Lagrange dual)**.** Consider the following optimization problem.

$$\begin{aligned} \min_{x,y\in\mathbb{R}} \quad & x - y \\ \text{s.t.} \quad & x^2 + y^2 = 1 \end{aligned}$$

The Lagrange dual for this problem is given as:

$$\sup_{\mu\in\mathbb{R}} \quad \inf_{x,y\in\mathbb{R}} \quad x - y - \mu \cdot (x^2 + y^2 - 1)$$

Instead of solving the original problem, we can now solve this simpler version, which is unconstrained.

In a first step, we compute the gradient and Hessian:

$$\nabla_{x,y}\mathcal{L}(\mu, x, y) = \begin{pmatrix} 1 - 2\mu \cdot x \\ -1 - 2\mu \cdot y \end{pmatrix} \qquad \text{and}$$
$$\nabla^2_{x,y}\mathcal{L}(\mu, x, y) = \begin{pmatrix} -2\mu & 0 \\ 0 & -2\mu \end{pmatrix}$$

By solving the equation $\nabla_{x,y}\mathcal{L}(\mu, x, y) = \vec{0}$ for $x$ and $y$ we obtain $x = \frac{1}{2\mu}$ and $y = -\frac{1}{2\mu}$. By plugging this result in turn into our original side constraints $x^2 + y^2 = 1$ we obtain

$$\left(\frac{1}{2\mu}\right)^2 + \left(-\frac{1}{2\mu}\right)^2 = 1 \quad \iff \quad \mu^2 = \frac{2}{2^2} \quad \iff \quad \mu = \pm\frac{1}{\sqrt{2}}$$

By inspecting the Hessian we see that only the solution $\mu = -\frac{1}{\sqrt{2}}$ yields a local minimum. Accordingly, we obtain the solution $x = -\frac{1}{\sqrt{2}}$ and $y = \frac{1}{\sqrt{2}}$ with objective function value $x - y = -\sqrt{2}$.

**Remark 48** (Interpretation as two-player game)**.** The intuition behind the Lagrange dual can be phrased as a game of yourself against a malicious opponent. In this game, you want that $\mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu})$ gets as small as possible and your opponent wants that it gets as big as possible. You have control over the variable $\vec{x}$ and your opponent over the variables $\vec{\lambda}$ and $\vec{\mu}$. If you violate any of the constraints, your opponent can punish you by setting the corresponding Lagrange multipliers to large values such that you do not achieve the low optimization value you hoped for. Therefore, to achieve the least possible value, you are not allowed to violate the constraints and thus the primal and duel problem correspond to each other.

However, as we will see in the next section, the Lagrange dual and the original problem are not *strictly* the same, although the difference is subtle.

### 1.4.2 *Duality Gaps*

**Remark 49** (Duality gaps in terms of the game metaphor)**.** To understand the subtle difference between dual and primal, let's return to the two-player game metaphor from Remark 48. We said that the Lagrange dual can be understood as a game where you control the variable $\vec{x}$ and wish to minimize the Lagrangian $\mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu})$ whereas your opponent controls the vairables $\vec{\lambda}$ and $\vec{\mu}$ and wants to *maximize* the Lagrangian $\mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu})$.

The subtle problem in this game is that the Lagrange dual assumes that *your opponent must move first*, whereas you can then choose $\vec{x}$ in perfect knowledge of your opponents choices. In this scenario, your opponent must try to forsee what you will choose and select their variables conservatively, thus potentially giving you more points than they'd like. In the primal problem, though, *you* must choose first and your opponent can punish you for wrong variable choices.

The difference in the outcome values of both versions of the game is called the *duality gap*.

**Definition 50** (Duality gap, weak duality, strong duality)**.** The difference $f^* - \mathcal{L}^*$ between the optimal value of a primal problem $f^*$ and its Lagrange dual $\mathcal{L}^*$ is called *duality gap*.

We say that *weak duality* holds for an optimization problem if the duality gap is non-negative.

We say that *strong duality* holds for an optimization problem if the duality gap is zero.

**Theorem 51** (Weak duality theorem)**.** *For every optimization problem, weak duality holds.*

*Proof.* We first prove a much more general result, namely the max-min inequality.

Let $\mathcal{X}$, $\mathcal{Y}$ be two arbitrary sets and let $\mathcal{L}$ be some function $\mathcal{L} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. Further, let $\phi$ be the function $\phi : \mathcal{X} \to \mathbb{R}$ with $\phi(x) := \inf_{y \in \mathcal{Y}} \mathcal{L}(x, y)$. Now, for any $x \in \mathcal{X}$, $\mathcal{L}(x, y)$ must be at least as big as $\phi(x)$, otherwise $\phi(x)$ would not be the infimum over all $y$. In other words, we obtain:

$$\mathcal{L}(x, y) \geqslant \phi(x) \quad \forall x \in \mathcal{X}, y \in \mathcal{Y}$$

Further, if we now maximize over $x$, $\sup_{x \in \mathcal{X}} \mathcal{L}(x, y)$ is at least as large as $\sup_{x \in \mathcal{X}} \phi(x)$. Otherwise $\phi(x)$ would, again, not be the infimum. Formally, we obtain:

$$\sup_{x \in \mathcal{X}} \quad \mathcal{L}(x, y) \geqslant \sup_{x \in \mathcal{X}} \phi(x) \quad y \in \mathcal{Y}$$

Finally, because this inequality holds for all $y \in \mathcal{Y}$, it also holds for the infimum, which yields the max-min inequality:

$$\inf_{y \in \mathcal{Y}} \quad \sup_{x \in \mathcal{X}} \quad \mathcal{L}(x, y) \geqslant \sup_{x \in \mathcal{X}} \quad \inf_{y \in \mathcal{Y}} \quad \mathcal{L}(x, y) \tag{1.38}$$

Next, let

$$\begin{aligned} \min_{\vec{x} \in \mathbb{R}^K} \quad & f(\vec{x}) \\ \text{s.t.} \quad & g_i(\vec{x}) \geqslant 0 && \forall i \in \{1, \dots, m\} \\ & h_j(\vec{x}) = 0 && \forall j \in \{1, \dots, n\} \end{aligned}$$

be an optimization problem in standard form. Then we can define the following problem:

$$\begin{aligned} \min_{\vec{x} \in \mathbb{R}^K} \quad & \phi(\vec{x}) && \text{where} \\ \phi(\vec{x}) = & \sup_{\vec{\lambda} \in \mathbb{R}^m_+, \vec{\mu} \in \mathbb{R}^n} \quad \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu}) \end{aligned}$$

and where $\mathcal{L}$ is the Lagrangian of the original problem (refer to Equation 1.37).

This problem is equivalent because for any $\vec{x}$ that is not in the feasible set, $\phi(\vec{x})$ obtains the value $\infty$, and for any $\vec{x}$ in the feasible set, $\phi(\vec{x}) = f(\vec{x})$. Therefore, the duality gap is exactly the difference between the optimal objective function values for our equivalent problem and the solution of the Lagrange dual. The optimal value of the equivalent problem is per definition above:

$$\inf_{\vec{x} \in \mathbb{R}^K} \quad \sup_{\vec{\lambda} \in \mathbb{R}^m_+, \vec{\mu} \in \mathbb{R}^n} \quad \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu})$$

whereas the optimal value of the Lagrange dual is per definition:

$$\sup_{\vec{\lambda} \in \mathbb{R}^m_+, \vec{\mu} \in \mathbb{R}^n} \quad \inf_{\vec{x} \in \mathbb{R}^K} \quad \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu})$$

Therefore, due to the max-min inequality 1.38, weak duality holds. □

**Example 52** (Strong duality counterexamples)**.** There exist optimization problems, even convex ones, where strong duality does *not* hold.

First, consider a simple non-convex example.

$$\begin{aligned} \min_{x,y\in\mathbb{R},y>0} \quad & x^2 \\ \text{s.t.} \quad & x\cdot y - y \geqslant 0 \end{aligned}$$

Note that this is equivalent to minimizing $x^2$ such that $x \geqslant 1$. The solution to this problem is thus obviously $x = 1$ with $f(1) = 1^2 = 1$. Now, consider the Lagrange dual.

$$\begin{aligned} \sup_{\lambda\in\mathbb{R}} \quad \inf_{x,y\in\mathbb{R},y>0} \quad & x^2 - \lambda\cdot(x\cdot y - y) \\ \text{s.t.} \quad & \lambda \geqslant 0 \end{aligned}$$

Because the inner optimization problem receives $\lambda$ as input, we can set $y = \frac{\epsilon}{\lambda}$ for arbitrarily $\epsilon > 0$, resulting in the Lagrangian $\mathcal{L}(x, \frac{\epsilon}{\lambda}, \lambda) = x^2 - \epsilon\cdot(x-1)$. This Lagrangian has the derivative $\frac{\partial}{\partial x}\mathcal{L}(x, \frac{\epsilon}{\lambda}, \lambda) = 2x - \epsilon$ and the second derivative $\frac{\partial^2}{\partial^2 x}\mathcal{L}(x, \frac{\epsilon}{\lambda}, \lambda) = 2$. Therefore, we obtain a minimum for $x = \frac{\epsilon}{2}$. Because we can set $\epsilon$ to arbitrarily small values larger than zero, we obtain an infimum of zero. Note that this infimum is independent of $\lambda$, such that our overall solution is 0 as well. Therefore, we obtain a duality gap of $1 - 0 = 1 > 0$.

Our next example is due to Tan (2015). Consider the problem

$$\begin{aligned} \min_{x,y\in\mathbb{R},y>0} \quad & \exp(-x) \\ \text{s.t.} \quad & -\frac{x^2}{y} \geqslant 0, \end{aligned}$$

which is convex. Note that the only possible feasible value for $x$ is 0 because otherwise the inequality constraint would be violated. Therefore, the minimal objective function value is 1.

Now, consider the Lagrange dual of this problem:

$$\begin{aligned} \sup_{\lambda\in\mathbb{R}} \quad \inf_{x,y\in\mathbb{R},y>0} \quad & \exp(-x) + \lambda\cdot\frac{x^2}{y} \\ \text{s.t.} \quad & \lambda \geqslant 0 \end{aligned}$$

Because the inner minimization problem has access to the value of $\lambda$, we can simply set $y = x^3\cdot\lambda$ and thus obtain the alternative objective function $\exp(-x) + \frac{1}{x}$, which is is minimized by setting $x$ to arbitrarily large values, thus achieving an infimum of zero. Note that this infimum is independent of $\lambda$, such that our overall solution is zero as well, yielding a duality gap of $1 - 0 = 1 > 0$.

The last example begs the question when strong duality *is* guaranteed. As it turns out, it is hard to characterize necessary conditions for strong duality. But we do now *sufficient* conditions, the broadest of which is Slater's condition.

**Definition 53** (Slater's condition)**.** Let

$$\begin{aligned} \min_{\vec{x}\in\mathbb{R}^K} \quad & f(\vec{x}) \\ \text{s.t.} \quad & g_i(\vec{x}) \geqslant 0 \qquad\qquad \forall i \in \{1,\dots,m\} \end{aligned}$$

$$\boldsymbol{A} \cdot \vec{x} = \vec{b}$$

be a convex optimization problem. We say that this problem conforms to *Slater's condition* if there exists at least one $\vec{x} \in \mathcal{X}^*$ such that $g_i(\vec{x}) > 0$ for all $i \in \{1, \dots, m\}$.

**Theorem 54** (Strong duality under Slater's condition)**.** *If an optimization problem fulfills Slater's condition, strong duality holds.*

*Proof.* This proof is, unfortunately, a little too involved to present here in full. We encourage the reader to inspect Boyd and Vandenberghe (2004, pp. 232-236) for a full proof. □

### 1.4.3 *Karush-Kuhn-Tucker conditions*

Strong duality tells us that we can solve the Lagrange dual instead of the primal problem to achieve a local minimum. The remaining question is how *do* we solve the Lagrange dual in general? In unconstrained optimization, we only had to care about the gradient being zero and the Hessian being positive definite. Now we also have to take constraints into account. The overall set of necessary constraints we need to fulfill for a local minimum of the Lagrange dual that may also be a local minimum of the primal problem are called the Karush-Kuhn-Tucker conditions.

**Definition 55** (Karush-Kuhn-Tucker conditions)**.** Let

$$\begin{aligned} \sup_{\vec{\lambda} \in \mathbb{R}^m, \vec{\mu} \in \mathbb{R}^n} \quad & \inf_{\vec{x} \in \mathbb{R}^K} \quad f(\vec{x}) - \sum_{i=1}^{m} \lambda_i \cdot g_i(\vec{x}) - \sum_{j=1}^{n} \mu_j \cdot h_j(\vec{x}) \\ & \text{s.t.} \quad \lambda_i \geqslant 0 \qquad\qquad \forall i \in \{1, \dots, m\} \end{aligned}$$

be the Lagrange dual of an optimization problem. Then, the Karush-Kuhn-Tucker conditions (KKTs) are defined as the following Equations.

$$g_i(\vec{x}) \geqslant 0 \qquad \forall i \in \{1, \dots, m\} \tag{1.39}$$

$$h_j(\vec{x}) = 0 \qquad \forall j \in \{1, \dots, n\} \tag{1.40}$$

$$\lambda_i \geqslant 0 \qquad \forall i \in \{1, \dots, m\} \tag{1.41}$$

$$\lambda_i \cdot g_i(\vec{x}) = 0 \qquad \forall i \in \{1, \dots, m\} \tag{1.42}$$

$$\nabla_{\vec{x}} f(\vec{x}) = \sum_{i=1}^{m} \lambda_i \cdot \nabla_{\vec{x}} g_i(\vec{x}) + \sum_{j=1}^{n} \mu_j \cdot \nabla_{\vec{x}} h_j(\vec{x}) \tag{1.43}$$

We call the set of inequality constraint for which $\lambda_i > 0$ the *active* inequality constraints.

**Remark 56** (Intuition behind the KKT conditions)**.** The first two KKT conditions ensure that $\vec{x}$ is still feasible point for the primal problem and the third condition ensures that $\vec{\lambda}$ is feasible for the Lagrange dual.

Regarding the fourth condition, consider the game metaphor again. If you manage to find a point where $g_i(\vec{x}) > 0$, i.e. the $i$th inequality constraint is strictly fulfilled, your opponent would only give you points if $\lambda_i > 0$. Therefore, they will set $\lambda_i = 0$ if $g_i(\vec{x}) > 0$. If $g_i(\vec{x}) \leqslant 0$ then we know that $g_i(\vec{x}) = 0$ because the first condition forbids $g_i(\vec{x}) < 0$. In conjunction, these two cases ensure that $\lambda_i \cdot g_i(\vec{x}) = 0$ for all $i$.

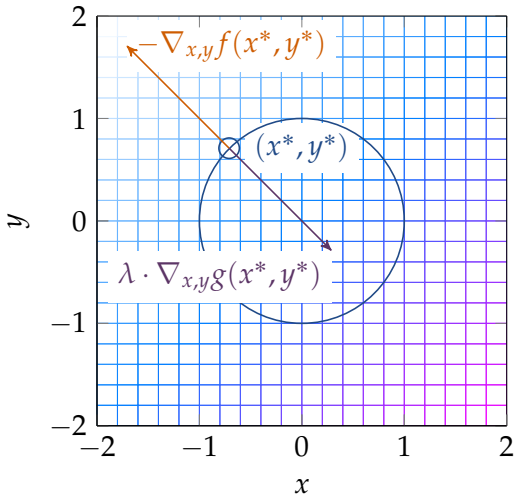



*Figure 1.7:* A geometric illustration for the fifth Karush-Kuhn-Tucker condition for the optimization problem $\min_{x,y} x - y$ such that $x^2 + y^2 \leqslant 1$ from Example 58. The mesh plot in the background illustrates the objective function $f(x, y) = x - y$ and the circle encloses the feasible set where $x^2 + y^2 \leqslant 1$. The only point which fulfills the fifth Karush-Kuhn-Tucker condition is $(x, y) = (-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ which is also the global minimum of the problem. At this point, the negative gradient of the objective function and the gradient of the ($\lambda$-weighted) inequality constraint function are exactly equally strong and thus cancel out.

The last condition is equivalent to saying that the gradient of the Lagrangian with respect to $\vec{x}$ should vanish. This requirement makes sense because we expect that $\vec{x}$ minimizes the Lagrangian due to the inner optimization problem of the Lagrange dual.

However, the last Karush-Kuhn-Tucker condition also has a nice and intuitive geometric interpretation: A local minimum of the problem must be at a point where the negative gradient of the objective function and the gradient of the constraints exactly cancel out (when weighted with the Lagrange multipliers). Figure 1.7 illustrates this geometric interpretation.

**Theorem 57** (Optimality under KKT conditions)**.** *The following claims hold:*

1. *If strong duality holds, then every point $(\vec{x}, \vec{\lambda}, \vec{\mu})$ that is a solution of the Lagrange dual where $\vec{x} \in \mathcal{X}^*$ fulfills the Karush-Kuhn-Tucker conditions.*

2. *Let $(\vec{x}, \vec{\lambda}, \vec{\mu})$ be a point which fulfills the Karush-Kuhn-Tucker conditions and let $\mathcal{S}(\vec{x}, \vec{\lambda})$ be the set of vectors which are orthogonal to gradients of the active inequality constraint and the gradients of all equality constraint, i.e.:*

$$\mathcal{S}(\vec{x}, \vec{\lambda}) := \big\{\vec{y} \in \mathbb{R}^K \big| \forall i \in \{1, \dots, m\} : \lambda_i \cdot (\nabla_{\vec{x}} g_i(\vec{x}))^T \cdot \vec{y} = 0, \\ \forall j \in \{1, \dots, n\} : (\nabla_{\vec{x}} h_j(\vec{x}))^T \cdot \vec{y} = 0\big\}$$

   *If strong duality holds, then every point $(\vec{x}, \vec{\lambda}, \vec{\mu})$ which fulfills the Karush-Kuhn-Tucker conditions and where $\vec{y}^T \cdot \nabla^2_{\vec{x}} \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu}) \cdot \vec{y} \geqslant 0$ for all $\vec{y} \in \mathcal{S}(\vec{x}, \vec{\lambda})$ is a local optimum of the Lagrange dual and $\vec{x}$ is a local minimum of the primal problem.*

3. *Let $(\vec{x}, \vec{\lambda}, \vec{\mu})$ be a point which fulfills the Karush-Kuhn-Tucker conditions. If the problem fulfills Slater's condition, then $(\vec{x}, \vec{\lambda}, \vec{\mu})$ is a global optimum of the Lagrange dual and $\vec{x}$ is a global minimum for the primal problem.*

*Proof.* Again, the full proof is too involved to present here. However, we can at least provide a proof for the first claim.

Let $(\vec{x}, \vec{\lambda}, \vec{\mu})$ be a solution of the Lagrange dual where $\vec{x} \in \mathcal{X}^*$. Because $\vec{x} \in \mathcal{X}^*$, the first two Karush-Kuhn-Tucker conditions are fulfilled. The third condition holds because $\vec{\lambda} \geqslant 0$ is a side constraint of the Lagrange dual. For the fourth and fifth condition, we need to go into more detail, though.

Assume that strong duality holds. Then, we know that

$$f(\vec{x}) = \inf_{\vec{x}'} \mathcal{L}(\vec{x}', \vec{\lambda}, \vec{\mu})$$

Further, because we consider the infimum here, we also obtain

$$\inf_{\vec{x}'} \mathcal{L}(\vec{x}', \vec{\lambda}, \vec{\mu}) \leqslant \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu})$$

Finally, because $\vec{x} \in \mathcal{X}^*$ and $\vec{\lambda} \geqslant 0$ we know that it holds:

$$\begin{aligned} -\lambda_i \cdot g_i(\vec{x}) &\leqslant 0 && \forall i \in \{1, \dots, m\} \\ -\mu_j \cdot h_j(\vec{x}) &= 0 && \forall j \in \{1, \dots, n\} \end{aligned}$$

Accordingly, we obtain:

$$f(\vec{x}) \geqslant f(\vec{x}) - \sum_{i=1}^{m} \lambda_i \cdot g_i(\vec{x}) - \sum_{j=1}^{n} \mu_j \cdot h_j(\vec{x}) = \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu})$$

which in turn implies that

$$f(\vec{x}) \leqslant \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu}) \leqslant f(\vec{x}) \quad \iff \quad f(\vec{x}) = \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu}).$$

Using this equality we can show the fourth and fifth Karush-Kuhn-Tucker conditions. In particular, because

$$f(\vec{x}) = \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu}) = f(\vec{x}) - \sum_{i=1}^{m} \lambda_i \cdot g_i(\vec{x})$$

we know that $\sum_{i=1}^{m} \lambda_i \cdot g_i(\vec{x}) = 0$. However, we also know that $\lambda_i \geqslant 0$ and $g_i(\vec{x}) \geqslant 0$ for all $i \in \{1, \dots, m\}$. Therefore, it must hold that $\lambda_i \cdot g_i(\vec{x}) = 0$ for all $i \in \{1, \dots, m\}$, which is precisely the fourth Karush-Kuhn-Tucker condition.

Further, because

$$\inf_{\vec{x}'} \mathcal{L}(\vec{x}', \vec{\lambda}, \vec{\mu}) = \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu})$$

we know that $\vec{x}$ is a minimum of the Lagrangian. Therefore, the gradient of the Lagrangian with respect to $\vec{x}$ must vanish, i.e.

$$\nabla_{\vec{x}} \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu}) = \nabla_{\vec{x}} f(\vec{x}) - \sum_{i=1}^{m} \lambda_i \cdot \nabla_{\vec{x}} g_i(\vec{x}) - \sum_{j=1}^{n} \mu_j \cdot \nabla_{\vec{x}} h_j(\vec{x}) = \vec{0}$$

which yields the final Karush-Kuhn-Tucker condition. □

**Example 58** (Solving a problem with the KKT conditions)**.** Consider the following problem.

$$\min_{x,y \in \mathbb{R}} \quad x - y$$

$$\text{s.t.} \quad x^2 + y^2 \leqslant 1 \iff 1 - x^2 - y^2 \geqslant 0$$

Note that this problem is convex and has no equality constraints. Further, consider the point $x = y = 0$. For this point, $0^2 + 0^2 = 0 < 1$, i.e. there exists a point where the inequality constraint is strictly fulfilled, which means that Slater's condition holds. Therefore, strong duality holds as well.

We obtain the following Karush-Kuhn-Tucker conditions.

$$\begin{aligned} 1 - x^2 - y^2 &\geqslant 0 \\ \lambda &\geqslant 0 \\ \lambda \cdot (1 - x^2 - y^2) &= 0 \\ \nabla_{x,y}(x - y) &= \lambda \cdot \nabla_{x,y}(1 - x^2 - y^2) \end{aligned}$$

Consider the last equation, which we can re-write as follows.

$$\begin{pmatrix} 1 \\ -1 \end{pmatrix} = \lambda \cdot \begin{pmatrix} -2x \\ -2y \end{pmatrix} \quad \iff \quad x = -\frac{1}{2\lambda} \wedge y = \frac{1}{2\lambda}$$

Plugging this result into the first equation we obtain:

$$1 - (-\frac{1}{2\lambda})^2 - (\frac{1}{2\lambda})^2 = 1 - \frac{1}{2\lambda^2} \geqslant 0 \quad \iff \quad \lambda^2 \geqslant \frac{1}{2} \quad \iff \quad \lambda \geqslant \frac{1}{\sqrt{2}} \vee \lambda \leqslant -\frac{1}{\sqrt{2}}$$

The latter option is impossible due to the condition $\lambda \geqslant 0$. Therefore, we obtain: $\lambda \geqslant \frac{1}{\sqrt{2}}$. Because $\lambda > 0$, the third equation implies that $1 - x^2 - y^2 = 0$, which in turn implies that $\lambda = \frac{1}{\sqrt{2}}$ and $x = -\frac{1}{\sqrt{2}}$ as well as $y = \frac{1}{\sqrt{2}}$. Because Slater's condition holds, Theorem 57 implies that this is a global minimum of our problem.

### 1.4.4 *Wolfe Dual Form*

Until now, we have first re-framed a problem in its Lagrange dual and then applied the Karush-Kuhn-Tucker conditions to solve it (provided that strong duality holds). It is also possible to plug in the key Karush-Kuhn-Tucker conditions directly into the Lagrange dual, which is then called the Wolfe dual.

**Definition 59** (Wolfe dual)**.** Let

$$\begin{aligned} \min_{\vec{x} \in \mathbb{R}^K} \quad & f(\vec{x}) \\ \text{s.t.} \quad & g_i(\vec{x}) \geqslant 0 && \forall i \in \{1, \dots, m\} \\ & h_j(\vec{x}) = 0 && \forall j \in \{1, \dots, n\} \end{aligned}$$

be an optimization problem in standard form. Then, we define the Wolfe dual of the problem as follows.

$$\begin{aligned} \sup_{\vec{x} \in \mathbb{R}^K, \vec{\lambda} \in \mathbb{R}^K, \vec{\mu} \in \mathbb{R}^K} \quad & \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu}) \\ \text{s.t.} \quad & \vec{\lambda} \geqslant 0 \\ & \nabla_{\vec{x}} f(\vec{x}) = \sum_{i=1}^{m} \lambda_i \cdot \nabla_{\vec{x}} g_i(\vec{x}) + \sum_{j=1}^{n} \mu_j \cdot \nabla_{\vec{x}} h_j(\vec{x}) \end{aligned}$$

**Remark 60** (Intuition of the Wolfe dual)**.** At first glance, it may be confusing that the Wolfe dual states that we *maximize* over $\vec{x}$ event though we minimized over $\vec{x}$ before. This works because the Wolfe dual is specifically intended for convex problems and we only consider points where the gradient of the Lagrangian with respect to $\vec{x}$ vanishes. In convex problems, those points are necessarily global minima, even if we try to maximize.

Another explanation is that the side constraint for the vanishing gradient implicitly couples $\vec{x}$ with $\vec{\lambda}$ and $\vec{\mu}$. Indeed, in many practical cases, we can provide an analytical expression for $\vec{x}$ in terms of $\vec{\lambda}$ and $\vec{\mu}$, such that we can remove $\vec{x}$ from our optimization problem entirely. In these cases, it is directly apparent that the maximization over $\vec{x}$ does not matter.

Why do we still maximize though, even if it does not matter? Because we can avoid the nested min/max structure of the Lagrange dual, which makes our problem easier.

**Theorem 61** (Weak duality of the Wolfe dual)**.** *For any convex optimization problem, the solution of the Wolfe dual is at most as large as the solution of the original problem.*

*Proof.* This proof is due to the original paper of Wolfe (1961). Let $\vec{x}^*$ be a solution of the original problem and let $(\vec{x}, \vec{\lambda}, \vec{\mu})$ be a solution of the Wolfe dual. Recall that we wish to prove that $f(\vec{x}^*) \geqslant \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu})$.

Now, because the objective function $f$ is convex we can use the first-order convexity condition from Theorem 41 to provide a lower bound for $f(\vec{x}^*)$, namely:

$$f(\vec{x}^*) - f(\vec{x}) \geqslant f(\vec{x}) + (\vec{x}^* - \vec{x})^T \cdot \nabla_{\vec{x}} f(\vec{x}) - f(\vec{x}) = (\vec{x}^* - \vec{x})^T \cdot \nabla_{\vec{x}} f(\vec{x})$$

Due to the side constraint of the Wolfe dual we know that $\nabla_{\vec{x}} f(\vec{x})$ must be equal to $\sum_{i=1}^m \lambda_i \cdot \nabla_{\vec{x}} g_i(\vec{x}) + \sum_{j=1}^n \mu_j \cdot \nabla_{\vec{x}} h_j(\vec{x})$. Accordingly, we obtain:

$$(\vec{x}^* - \vec{x})^T \cdot \nabla_{\vec{x}} f(\vec{x}) = \sum_{i=1}^m \lambda_i \cdot (\vec{x}^* - \vec{x})^T \cdot \nabla_{\vec{x}} g_i(\vec{x}) + \sum_{j=1}^n \mu_j \cdot (\vec{x}^* - \vec{x})^T \cdot \nabla_{\vec{x}} h_j(\vec{x})$$

Because $-g_i$ is convex for all $i$ and $-h_j$ is convex for all $j$, we also obtain:

$$\begin{aligned} -g_i(\vec{x}^*) \geqslant -g_i(\vec{x}) - (\vec{x}^* - \vec{x}) \cdot \nabla_{\vec{x}} g_i(\vec{x}) &\iff (\vec{x}^* - \vec{x}) \cdot \nabla_{\vec{x}} g_i(\vec{x}) \geqslant g_i(\vec{x}^*) - g_i(\vec{x}) \quad \text{and} \\ -h_j(\vec{x}^*) \geqslant -h_j(\vec{x}) - (\vec{x}^* - \vec{x}) \cdot \nabla_{\vec{x}} h_j(\vec{x}) &\iff (\vec{x}^* - \vec{x}) \cdot \nabla_{\vec{x}} h_j(\vec{x}) \geqslant h_j(\vec{x}^*) - h_j(\vec{x}) \end{aligned}$$

for all $i \in \{1, \ldots, m\}$ and all $j \in \{1, \ldots, n\}$.

Therefore, we can conclude:

$$\begin{aligned} &\sum_{i=1}^m \lambda_i \cdot (\vec{x}^* - \vec{x})^T \cdot \nabla_{\vec{x}} g_i(\vec{x}) + \sum_{j=1}^n \mu_j \cdot (\vec{x}^* - \vec{x})^T \cdot \nabla_{\vec{x}} h_j(\vec{x}) \\ \geqslant &\sum_{i=1}^m \lambda_i \cdot \big(g_i(\vec{x}^*) - g_i(\vec{x})\big) + \sum_{j=1}^n \mu_j \cdot \big(h_j(\vec{x}^*) - h_j(\vec{x})\big) \end{aligned}$$

Because $\vec{x}^*$ is a feasible point of the primal problem, it must hold that $g_i(\vec{x}^*) \geqslant 0$ and that $h_j(\vec{x}^*) = 0$. Further, because $(\vec{x}, \vec{\lambda}, \vec{\mu})$ is a feasible point for the Wolfe dual, it must hold that $\lambda_i \geqslant 0$ for all $i$. Therefore, we can conclude:

$$\sum_{i=1}^m \lambda_i \cdot \big(g_i(\vec{x}^*) - g_i(\vec{x})\big) + \sum_{j=1}^n \mu_j \cdot \big(h_j(\vec{x}^*) - h_j(\vec{x})\big) \geqslant -\sum_{i=1}^m \lambda_i \cdot g_i(\vec{x}) - \sum_{j=1}^n \mu_j \cdot h_j(\vec{x})$$

Therefore, we obtain overall:

$$f(\vec{x}^*) - f(\vec{x}) \geqslant -\sum_{i=1}^{m} \lambda_i \cdot g_i(\vec{x}) - \sum_{j=1}^{n} \mu_j \cdot h_j(\vec{x}) \quad \iff \quad f(\vec{x}^*) \geqslant \mathcal{L}(\vec{x}, \vec{\lambda}, \vec{\mu}),$$

which concludes the proof. □

**Theorem 62** (Strong duality of the Wolfe dual)**.** *If an optimization problem fulfills Slater's condition then the duality gap between the primal and the Wolfe dual is zero.*

*Proof.* Refer to Wolfe (1961) for the forward direction. The backward direction holds because any point $\vec{x}$ where the gradient of the Lagrangian vanishes is a global minimum in case of convexity. Therefore, $(\vec{x}, \vec{\lambda}, \vec{\mu})$ is a solution of the Lagrange dual, which according to Theorem 54 implies a solution of the primal problem. □

**Example 63** (Wolfe dual)**.** Consider again the problem from Example 58

$$\begin{aligned} \min_{x,y \in \mathbb{R}} \quad & x - y \\ \text{s.t.} \quad & x^2 + y^2 \leqslant 1 \iff 1 - x^2 - y^2 \geqslant 0 \end{aligned}$$

As stated above, this problem fulfills Slater's condition.

We obtain the following Wolfe dual.

$$\begin{aligned} \sup_{x,y,\lambda \in \mathbb{R}} \quad & x - y - \lambda \cdot (1 - x^2 - y^2) \\ \text{s.t.} \quad & \lambda \geqslant 0 \\ & \nabla_{x,y}(x - y) = \lambda \cdot \nabla_{x,y}(1 - x^2 - y^2) \end{aligned}$$

As before, the last equation yields $x = -\frac{1}{2\lambda}$ and $y = \frac{1}{2\lambda}$. If we plug this into our objective function we obtain:

$$\begin{aligned} \mathcal{L}(-\frac{1}{2\lambda}, \frac{1}{2\lambda}, \lambda) &= -\frac{1}{\lambda} - \lambda \cdot \left(1 - (-\frac{1}{2\lambda})^2 - (\frac{1}{2\lambda})^2\right) \\ &= -\frac{1}{\lambda} - \lambda + \lambda \cdot \frac{1}{2\lambda^2} \\ &= -\frac{1}{2\lambda} - \lambda \end{aligned}$$

Accordingly, we can re-write our Wolfe dual as follows.

$$\begin{aligned} \inf_{\lambda \in \mathbb{R}} \quad & \frac{1}{2\lambda} + \lambda \\ \text{s.t.} \quad & \lambda \geqslant 0 \end{aligned}$$

Note that our original variables $x$ and $y$ have vanished from this problem. This is a typical phenomenon in the Wolfe dual.

By setting the derivative of the new objective function to zero we obtain:

$$\frac{\partial}{\partial \lambda}(\frac{1}{2\lambda} + \lambda) = -\frac{1}{2\lambda^2} + 1 = 0 \quad \iff \quad \lambda^2 = \frac{1}{2} \quad \iff \quad \lambda = \pm\frac{1}{\sqrt{2}}$$

We can exclude the possibility of $\lambda$ being negative due to our side constraint $\lambda \geqslant 0$. Accordingly, we obtain $\lambda = \frac{1}{\sqrt{2}}$, $x = -\frac{1}{\sqrt{2}}$ and $y = \frac{1}{\sqrt{2}}$. This is a global minimum due to Slater's condition.

# 2 ALGORITHMS

In this chapter, we cover methods to solve optimization problems. These methods come in three distinct flavors. First, we consider *analytical methods*, which employ your brain, pencil, paper, and symbols in order to derive a closed-form expression for a solution – basically, what you learned in school, but a bit cooler. Second, we consider *numerical methods*, which can get arbitrarily close to a local minimum in a differentiable optimization problem by means of some iterative, numeric algorithm. Finally, we consider *heuristics* which we have to use whenever the first two approaches fail. These heuristics can seldomly guarantee convergence but work even if no gradient information is available or when the problem is discrete.

In addition to these three general classes, we consider two important special cases of numeric optimization, namely probabilistic optimization and convex programming. In both cases, we can utilize special properties of the domain at hand to facilitate faster solutions than would be possible with generic numeric approaches.

## 2.1 ANALYTICAL METHODS

If we intend to solve an optimization problem analytically, we always follow the same basic scheme: First, write down the problem in a way that facilitates analytic optimization. Second, set the gradient to zero and solve for the variables of interest. Third, inspect the Hessian to ensure that we actually found a local minimum or even an global minimum. Note that this general scheme *always* requires that our problem is *continuous* and that our objective function and our constraint functions are twice differentiable. However, there are important differences depending on whether our problem is unconstrained, equality-constrained, or inequality-constrained. We will now cover these three settings in turn, providing a detailed recipe for optimization and an example in each case.

### 2.1.1 *Unconstrained Optimization*

Assume a problem of the form

$$\min_{\vec{x} \in \mathbb{R}^K} \quad f(\vec{x})$$

where $f$ is twice-differentiable. Then, we can employ the following recipe for optimization.

1. Compute the gradient $\nabla_{\vec{x}} f(\vec{x})$.
2. Solve the equation $\nabla_{\vec{x}} f(\vec{x}) = \vec{0}$ for $\vec{x}$. Let's call the solution $\vec{x}^*$.
3. If you already know that $f$ is convex, you are finished, because in this case Theorem 41 implies that the Hessian at every location is positive semi-definite, which implies that $\vec{x}^*$ is a local minimum (also refer to Theorem 31), and according to Theorem 40 every local minimum in a convex problem is a global minimum.
4. If you do *not* know that $f$ is convex, compute the Hessian $\nabla^2_{\vec{x}} f(\vec{x}^*)$.

5. Then compute the eigenvalues of $\nabla^2_{\vec{x}} f(\vec{x}^*)$.
6. If all eigenvalues are positive, you obtained a local minimum, because then Theorem 35 implies that the Hessian is positive definite and Theorem 31 implies that $\vec{x}^*$ is a local minimum. If at least one eigenvalue is negative, you did definitely *not* find a local minimum and need to find another solution $\vec{x}^*$ (also refer to Theorem 31). If no eigenvalue is negative but at least one is zero, you may have found an local minimum, but you can't be sure (also refer to Theorem 31).
7. To verify whether your solution is also a global minimum, there are multiple options. If all eigenvalues of the Hessian are positive *independent of* $\vec{x}^*$, then Theorem 41 implies that $f$ is convex in which case $\vec{x}^*$ is a global minimum according to Theorem 40. Otherwise, you can inspect all other solutions where the gradient is zero *and* all boundaries of the feasible set and compare your objective function value $f(\vec{x}^*)$ against the objective function values in these cases.

**Example 64** (Clear global minimum)**.** First, consider the problem

$$\min_{x,y\in\mathbb{R}} \quad y\cdot(3x+3y-9)+x\cdot(2x-7)+8$$

1. We obtain the gradient
$$\nabla_{(x,y)} f(x,y) = \begin{pmatrix} 3y+4x-7 \\ 3x+6y-9 \end{pmatrix}$$
2. Setting the gradient to zero yields:
$$\begin{aligned} 3y^*+4x^*-7=0 &\iff 3y^*=7-4x^* \\ \Rightarrow 3x^*+2\cdot(7-4x^*)-9=0 &\iff -5x^*+5=0 \iff x^*=1 \end{aligned}$$
Plugging this result into the first equation yields: $3y^*=7-4 \iff y^*=1$.
3. We do not yet know whether $f$ is convex.
4. We obtain the Hessian:
$$\nabla^2_{(x,y)} f(x^*,y^*) = \begin{pmatrix} 4 & 3 \\ 3 & 6 \end{pmatrix}$$
5. The eigenvalues of the Hessian are $\lambda_1 = 5+\sqrt{10}$ and $\lambda_2 = 5-\sqrt{10}$.
6. Both eigenvalues are positive. Therefore, the Hessian is positive definite and $(x^*,y^*)=(1,1)$ is a local minimum.
7. Further note that our Hessian is a constant that is *independent* of our solution $(x^*,y^*)$. Therefore, $f$ is convex and $(x^*,y^*)=(1,1)$ is even a global minimum.

**Example 65** (Global minimum by convexity)**.** Next, consider the problem

$$\min_{x,y\in\mathbb{R}} (x+y-1)^2$$

1. We obtain the gradient
$$\nabla_{(x,y)} f(x,y) = \begin{pmatrix} 2\cdot(x+y-1) & 2\cdot(x+y-1) \end{pmatrix}$$

2. Setting the gradient to zero yields:

$$2 \cdot (x^* + y^* - 1) = 0 \quad \iff \quad y^* = 1 - x^*$$

So we have infinitely many solutions $(x^*, y^*)$.

3. We do not yet know whether $f$ is convex.

4. We obtain the Hessian:

$$\nabla^2_{(x,y)} f(x^*, y^*) = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

5. The eigenvalues of the Hessian are $\lambda_1 = 4$ and $\lambda_2 = 0$.

6. One eigenvalue is positive and one is zero, therefore our Hessian is positive semi-definite but not positive definite and we do not know yet whether we have obtained a local minimum

7. However, note that our Hessian is a constant that is *independent* of our solution $(x^*, y^*)$. Therefore, $f$ is convex and every solution $(x^*, y^*)$ with $y^* = 1 - x^*$ is a global minimum.

**Example 66** (Local, but not global minimum)**.** Finally, consider the problem

$$\min_{x \in \mathbb{R}} x + (1 - x)^3$$

1. We obtain the gradient

$$\nabla_x f(x) = 1 - 3 \cdot (1 - x)^2$$

2. Setting the gradient to zero yields:

$$1 - 3 \cdot (1 - x^*)^2 = 0 \quad \iff \quad x_1^* = 1 + \frac{1}{\sqrt{3}} \vee x_2^* = 1 - \frac{1}{\sqrt{3}}$$

So we have two solutions for $x^*$.

3. We do not yet know whether $f$ is convex.

4. We obtain the Hessian:

$$\nabla^2_x f(x^*) = 6 \cdot (1 - x^*)$$

5. The eigenvalue of the Hessian for the first solution $x_1^* = 1 + \frac{1}{\sqrt{3}}$ is $6 \cdot (1 - 1 - \frac{1}{\sqrt{3}}) = -\frac{6}{\sqrt{3}} < 0$. The eigenvalue of the Hessian for the second solution $x_2^* = 1 - \frac{1}{\sqrt{3}}$ is $6 \cdot (1 - 1 + \frac{1}{\sqrt{3}}) = \frac{6}{\sqrt{3}} > 0$.

6. Our first solution is not a local minimum, but our second solution is, because the single eigenvalue is positive.

7. Our Hessian is *not* a constant. Therefore, we have to inspect the objective function values at the boundaries and compare them with our solution. For our solution we obtain $f(x_2^*) = 1 - \frac{1}{\sqrt{3}} + \frac{1}{(\sqrt{3})^3}$. Unfortunately, for any $x > 1 + \frac{2}{\sqrt{3}}$ we obtain smaller objective function values. So we found a local minimum but *not* a global minimum, which instead lies at $\infty$.

### 2.1.2 *Equality-Constrained Optimization*

Assume a problem of the form:

$$\begin{aligned} \min_{\vec{x}\in\mathbb{R}^K} \quad & f(\vec{x}) \\ \text{s.t.} \quad & h_j(\vec{x}) = 0 \qquad\qquad \forall j \in \{1,\dots,n\} \end{aligned}$$

where $f$ and all $h_j$ are twice differentiable. Then, you can apply the following recipe.

1. Try to solve the equality constraints for as many variables $x_k$ as possible. Then, remove these variables and equality constraints from the optimization problem. If *all* equality constraints can be removed in this way, your optimization problem becomes unconstrained and you can continue with the recipe for unconstrained optimization. Otherwise, continue here.

2. For the remaining problem, construct the Lagrange dual:

$$\sup_{\vec{\mu}\in\mathbb{R}^n} \quad \inf_{\vec{x}\in\mathbb{R}^k} \quad \mathcal{L}(\vec{x},\vec{\mu}) \qquad\qquad \text{where}$$

$$\mathcal{L}(\vec{x},\vec{\mu}) = f(\vec{x}) - \sum_{j=1}^{n} \mu_j \cdot h_j(\vec{x})$$

   Note that this problem is unconstrained.

3. Compute the gradient $\nabla_{\vec{x}}\mathcal{L}(\vec{x},\vec{\mu})$.

4. Solve the equation $\nabla_{\vec{x}}\mathcal{L}(\vec{x},\vec{\mu}) = \vec{0}$ for $\vec{x}$. Let's call the solution $\vec{x}^*$.

5. Consider the equations $h_j(\vec{x}^*) = 0$ for all $j \in \{1,\dots,m\}$ and solve for $\mu_1,\dots,\mu_m$. Let's call the solution $\vec{\mu}^*$.

6. Plug $\vec{\mu}^*$ in your solution $\vec{x}^*$.

7. If you know that your problem fulfills Slater's condition, then you are finished because then Theorem 57 implies that $\vec{x}^*$ is a global minimum of the primal problem. If you do not yet know whether your problem fulfills Slater's condition, check whether all your equality constraints functions are affine and whether your objective function is convex. If that is the case, Slater's condition is fulfilled. Otherwise, it is not.

8. Otherwise, compute the Hessian $\nabla^2_{\vec{x}}\mathcal{L}(\vec{x}^*,\vec{\mu}^*)$.

9. Compute the eigenvalues of $\nabla^2_{\vec{x}}\mathcal{L}(\vec{x}^*,\vec{\mu}^*)$.

10. If all eigenvalues are positive, then Theorem 41 implies that the Hessian is positive definite and Theorem 57 implies that $\vec{x}^*$ is a local minimum of the primal problem - under the assumption of strict duality. In other cases you cannot be sure whether you found a local minimum. You can, however, try to consider only vectors which are orthogonal to the active constraint gradient directions and prove positive definiteness of the Hessian for those, which is also a sufficient condition.

11. To verify whether your solution is also a global minimum, you need to consider all other possible solutions of your dual as well as infinity boundaries and compare the objective function value $f(\vec{x}^*)$ against the objective function values in these cases.

12. Recover your full solution $\vec{x}^*$ by plugging your solution for the reduced problem into the equality constraints that you used to reduce the problem in the beginning.

**Remark 67** (Usefulness of Slater's condition)**.** In analytical equality-constrained optimization, Slater's condition is highly useful, because it is relatively easy to prove and frees you from steps 8-12 in the recipe above. So it is highly recommended to check Slater's condition first.

**Example 68** (Solution by problem reduction)**.** First, consider the problem

$$\begin{aligned} \min_{x,y\in\mathbb{R}} \quad & x^2 + \frac{1}{2}y^2 \\ \text{s.t.} \quad & x + y = 1 \end{aligned}$$

1. We can solve the equality constraint $x + y = 1$ for $y$ via $y = 1 - x$. Accordingly, we obtain the following alternative problem:

$$\min_{x\in\mathbb{R}} \quad x^2 + \frac{1}{2}(1-x)^2$$

Note that this problem is unconstrained. Therefore, we can continue with the recipe for unconstrained optimization.

a) We obtain the gradient

$$\nabla_x x^2 + \frac{1}{2}(1-x)^2 = 2x - (1-x) = 3x - 1$$

b) By setting the gradient to zero we obtain

$$3x^* - 1 = 0 \quad \iff \quad x^* = \frac{1}{3}$$

c) We do not yet know whether our objective function is convex.

d) We obtain the Hessian

$$\nabla_x^2 x^2 + \frac{1}{2}(1-x)^2 = 3$$

e) The single eigenvalue is 3.

f) This eigenvalue is positive, therefore $x^* = \frac{1}{3}$ is a local minimum.

g) Our Hessian is a positive definite constant, therefore $f$ is convex and $x^* = \frac{1}{3}$ is a global minimum.

2. Via our equality constraint we recover $y^* = 1 - x^* = \frac{2}{3}$, implying that $(x^*, y^*) = (\frac{1}{3}, \frac{2}{3})$ is a global minimum of the original problem.

**Example 69** (Quadratic Programming / Solution by Lagrange Dual)**.** Consider a general, linearly inequality-constrained quadratic program of the following form:

$$\begin{aligned} \min_{\vec{x}\in\mathbb{R}^K} \quad & \frac{1}{2}\vec{x}^T \cdot \boldsymbol{P} \cdot \vec{x} + \vec{q}^T \cdot \vec{x} \\ \text{s.t.} \quad & \boldsymbol{A} \cdot \vec{x} = \vec{b} \end{aligned}$$

where $\boldsymbol{P}$ is a symmetric and positive definite $K \times K$ matrix, where $\vec{q}$ is a $K$-dimensional vector, where $\boldsymbol{A}$ is a $n \times K$ matrix, and where $\vec{b}$ is a $n$-dimensional vector.

1. Because $\boldsymbol{A}$ is generally not invertible, we can not solve for $\vec{x}$. So we leave the problem as is.

2. We obtain the following Lagrange dual.

$$\sup_{\vec{\mu}\in\mathbb{R}^n} \quad \inf_{\vec{x}\in\mathbb{R}^K} \quad \mathcal{L}(\vec{x},\vec{\mu}) \qquad \text{where}$$
$$\mathcal{L}(\vec{x},\vec{\mu}) = \vec{x}^T \cdot \boldsymbol{P} \cdot \vec{x} + \vec{q}^T \cdot \vec{x} - \vec{\mu}^T \cdot (\boldsymbol{A} \cdot \vec{x} - \vec{b})$$

3. We obtain the following gradient of the Lagrangian:

$$\nabla_{\vec{x}} \mathcal{L}(\vec{x},\vec{\mu}) = \boldsymbol{P} \cdot \vec{x} + \vec{q} - \boldsymbol{A}^T \cdot \vec{\mu}$$

4. By setting the gradient to zero, we obtain the following solution $\vec{x}^*$:

$$\nabla_{\vec{x}} \mathcal{L}(\vec{x},\vec{\mu}) = \vec{0} \quad \iff \quad \boldsymbol{P} \cdot \vec{x} = \boldsymbol{A}^T \cdot \vec{\mu} - \vec{q} \quad \iff \quad \vec{x} = \boldsymbol{P}^{-1} \cdot \left(\boldsymbol{A}^T \cdot \vec{\mu} - \vec{q}\right)$$

   Note that the last step is valid because $\boldsymbol{P}$ is positive definite and therefore invertible.

5. By plugging this result into our equality constraints we obtain the following result for $\vec{\mu}^*$:

$$\begin{aligned} & \boldsymbol{A} \cdot \boldsymbol{P}^{-1} \cdot \left(\boldsymbol{A}^T \cdot \vec{\mu} - \vec{q}\right) = \vec{b} \\ \iff & \boldsymbol{A} \cdot \boldsymbol{P}^{-1} \cdot \boldsymbol{A}^T \cdot \vec{\mu} = \vec{b} + A \cdot \boldsymbol{P}^{-1} \cdot \vec{q} \\ \iff & \vec{\mu} = \left(\boldsymbol{A} \cdot \boldsymbol{P}^{-1} \cdot \boldsymbol{A}^T\right)^{-1} \cdot \left(\vec{b} + A \cdot \boldsymbol{P}^{-1} \cdot \vec{q}\right) \end{aligned}$$

   Note that $\boldsymbol{A} \cdot \boldsymbol{P}^{-1} \cdot \boldsymbol{A}^T$ is a symmetric square matrix which is either full rank or can be made full-rank by removing redundant equality constraints. Therefore, this matrix is invertible and the last step is valid.

6. By plugging this result into our solution $\vec{x}^*$ we obtain:

$$\vec{x}^* = \boldsymbol{P}^{-1} \cdot \left(\boldsymbol{A}^T \cdot \left(\boldsymbol{A} \cdot \boldsymbol{P}^{-1} \cdot \boldsymbol{A}^T\right)^{-1} \cdot \left(\vec{b} + A \cdot \boldsymbol{P}^{-1} \cdot \vec{q}\right) - \vec{q}\right)$$

7. We do not know yet if our problem fulfills Slater's condition. But we notice that all our equality constraints functions are affine. Further, we obtain the following Hessian of our objective function:

$$\nabla^2_{\vec{x}} f(\vec{x}) = \boldsymbol{P}$$

   This is a positive definite constant. Therefore, our objective function is convex and Slater's condition is fulfilled. This further implies that $\vec{x}^*$ is a global minimum.

### 2.1.3 *Inequality-Constrained Optimization*

Assume a problem of the form:

$$\begin{aligned} \min_{\vec{x}\in\mathbb{R}^K} \quad & f(\vec{x}) \\ \text{s.t.} \quad & g_i(\vec{x}) \geqslant 0 && \forall i \in \{1,\dots,m\} \\ & h_j(\vec{x}) = 0 && \forall j \in \{1,\dots,n\} \end{aligned}$$

where $f$, all $g_i$, and all $h_j$ are twice differentiable.

These problems are *not* generally feasible for analytic optimization. It is only possible to 'trick' by a simple observation: Either, inequality constraints are *exactly* fulfilled, in which case they actually play the role of equality constraints, or they are strictly fulfilled, in which case we can ignore them. Therefore, the trick to solve an inequality-constrained problem analytically is to guess which constraints are 'relevant' and to incorporate those as equality constraints. Then, you can use the recipe above for this alternative problem.

**Example 70** (Tin Can Solution)**.** Consider the tin-can optimization problem from Example 3.

$$\begin{aligned} \min_{(r,h)\in\mathbb{R}^2} \quad & 2\pi \cdot r \cdot h + 2\pi \cdot r^2 \\ \text{s.t.} \quad & \pi \cdot r^2 \cdot h \geqslant 1500 \\ & r \geqslant 0 \\ & h \geqslant 0 \end{aligned}$$

If we would ignore all of our inequality constraints, we could, for example, set $r = 1$ and $h \to -\infty$ and achieve an arbitrarily good objective function value. However, this would violate both our first and third inequality constraint. Therefore, we need to incorporate at least one of those. Note, however, that if the first constraint is fulfilled, this implies that the third constraint is also fulfilled, because $\pi \cdot r^2$ is positive and if $h < 0$ than $\pi \cdot r^2 \cdot h < 0 < 1500$. Therefore, our guess is that we can ignore the second and third constraint, but not the first. We obtain the following alternative problem:

$$\begin{aligned} \min_{(r,h)\in\mathbb{R}^2} \quad & 2\pi \cdot r \cdot h + 2\pi \cdot r^2 \\ \text{s.t.} \quad & \pi \cdot r^2 \cdot h = 1500 \end{aligned}$$

Now, we can employ equality-constrained optimization

1. We first note that we can solve our equality constraint for $h$, i.e.:

$$\pi \cdot r^2 \cdot h = 1500 \quad \Longleftrightarrow \quad h = \frac{1500}{\pi \cdot r^2}$$

Therefore, we obtain the reduced problem:

$$\min_{r\in\mathbb{R}} \quad 2\pi \cdot r \cdot \frac{1500}{\pi \cdot r^2} + 2\pi \cdot r^2 = \frac{3000}{r} + 2\pi \cdot r^2$$

which is unconstrained. Therefore, we can continue with unconstrained optimization.

a) We obtain the gradient:

$$\nabla_r \frac{3000}{r} + 2\pi \cdot r^2 = -\frac{3000}{r^2} + 4\pi r$$

b) By setting the gradient to zero we obtain:

$$-\frac{3000}{(r^*)^2} + 4\pi r^* = 0 \quad \Longleftrightarrow \quad 4\pi (r^*)^3 = 3000 \quad \Longleftrightarrow \quad r^* = \sqrt[3]{\frac{750}{\pi}}$$

c) We do not yet know whether $f$ is convex.

d) We obtain the Hessian:

$$\nabla_r^2 \frac{3000}{r^*} + 2\pi \cdot (r^*)^2 = \frac{6000}{(r^*)^3} + 4\pi = 8\pi + 4\pi = 12\pi$$

e) The single eigenvalue $12\pi$ is positive. Therefore, $r^*$ is a local minimum.

f) To verify that we also obtained a global minimum, we need to inspect the boundary cases. For $r \to 0$ we obtain:

$$\lim_{r \to 0} f(r) = \lim_{r \to 0} \frac{3000}{r} + 2\pi \cdot r^2 = \infty + 0$$

And for $r \to \infty$ we obtain:

$$\lim_{r \to \infty} f(r) = \lim_{r \to \infty} \frac{3000}{r} + 2\pi \cdot r^2 = 0 + \infty$$

In both cases, our objective function goes to infinity, which means that we did indeed obtain a global minimum.

2. We recover

$$h^* = \frac{1500}{\pi \cdot r^2} = \frac{1500}{\pi} \cdot \sqrt[3]{\frac{\pi^2}{750^2}} = \sqrt[3]{\frac{6000}{\pi}}$$

## 2.2 NUMERIC METHODS

In this section we cover *numeric* methods, by which we mean algorithms which can be executed by a computer to find a point $\vec{x}^* \in \mathcal{X}^*$ that is arbitrarily close to a local minimum of the problem. We first consider methods for unconstrained optimization and then continue to constrained optimization. Note that all methods in this section still assume that we have a continuous optimization problem with differentiable, ideally even convex objective function and constraint functions.

### 2.2.1 *Unconstrained Optimization*

In this section, we consider optimization problems of the form

$$\min_{\vec{x} \in \mathbb{R}^K} \quad f(\vec{x})$$

where $f$ is twice-differentiable.

The basic scheme of unconstrained numeric optimization is to start from some initial point $\vec{x}_0 \in \mathbb{R}^K$, then select some direction $\vec{p}_1 \in \mathbb{R}^K$ such that the objective function gets smaller in that direction, then select some $\alpha_1 \in \mathbb{R}_+$ (the *step size*) and set $\vec{x}_1 \leftarrow \vec{x}_0 + \alpha_1 \cdot \vec{p}_1$. This scheme is iterated until some stopping criterion is fulfilled, for example if the objective function drops below a certain value or if the norm of the gradient gets small.

#### Gradient Descent

Gradient descent is the most basic version of the iterative optimization scheme sketched above. We always select the *negative gradient* as our search direction, use a constant step size $\alpha$, and stop optimization as soon as the norm of our gradient drops below a threshold $\epsilon > 0$. The algorithm is displayed in Algorithm 1.

**Algorithm 1** Gradient descent for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, a starting value $\vec{x}_0 \in \mathbb{R}^K$, a step size $\alpha > 0$, and a gradient threshold $\epsilon > 0$.

**function** GRADIENT DESCENT(objective function $f : \mathbb{R}^K \to \mathbb{R}$, starting value $\vec{x}_0 \in \mathbb{R}^K$, step size $\alpha > 0$, threshold $\epsilon > 0$)
    $t \leftarrow 0$.
    **while** $\|\nabla_{\vec{x}} f(\vec{x}_t)\| > \epsilon$ **do**
        $\vec{x}_{t+1} \leftarrow \vec{x}_t - \alpha \cdot \nabla_{\vec{x}} f(\vec{x}_t)$.
        $t \leftarrow t + 1$.
    **end while**
    **return** $\vec{x}_t$.
**end function**

This makes intuitive sense: The gradient tells us in which direction the objective function increases the most, so the negative gradient tells us in which direction it *decreases* the most. Further, if our gradient gets small then we are close either to a local minimum or to a saddle point. If our problem is convex, we don't even have saddle points, so we are guaranteed to find a local minimum, which is in turn guaranteed to be a global minimum. The worst thing that could happen is that we always 'jump over' the actual local minimum because our step size is too large. So we need to set our step size small enough.

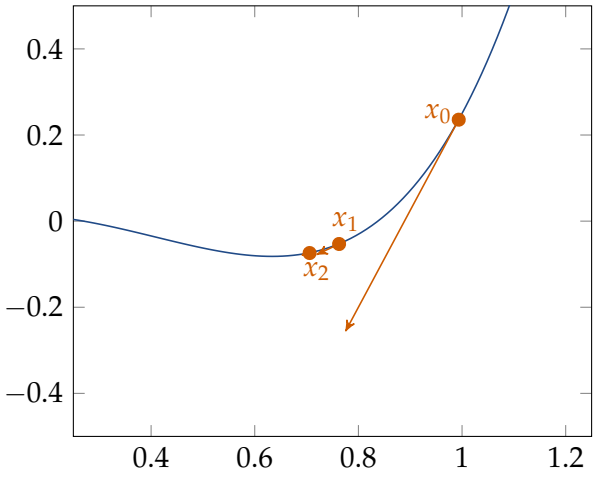



*Figure 2.1:* An illustration of gradient descent on the function $f(x) = x^4 - x^2 + \frac{1}{4}x$, starting at position $x_0 = 1$ with step size $\alpha = 0.1$. The arrows indicate the direction vector, i.e. $-\alpha \cdot \nabla_x f(x_t)$.

This intuitive explanation is good enough for all practical intents and purposes. However, if we look deeper, we can also justify gradient descent from a theoretical point of view. In particular, we can ask ourselves: What is the best next point $\vec{x}_{t+1}$ we could go if we wish to reduce the first-order Taylor approximation of our objective function around our current point $\vec{x}_t$ the most? Recall that the first-order Taylor approximation of our objective function around $\vec{x}_t$ is given as follows.

$$\tilde{f}^1_{\vec{x}_t}(\vec{x}_{t+1}) = f(\vec{x}_t) + (\vec{x}_{t+1} - \vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t)$$

So minimizing this approximation corresponds to the following optimization problem.

$$\min_{\vec{x}_{t+1} \in \mathbb{R}^K} \quad f(\vec{x}_t) + (\vec{x}_{t+1} - \vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t)$$

The issue with this optimization problem is that it is unbounded (as long as $\nabla_{\vec{x}} f(\vec{x}_t) \neq \vec{0}$). To see this, imagine that we set $\vec{x}_{t+1} = \vec{x}_t - \alpha \cdot \nabla_{\vec{x}} f(\vec{x}_t)$. Our optimization problem then reduces to:

$$\min_{\alpha \in \mathbb{R}} \quad f(\vec{x}_t) - \alpha \cdot \nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t)$$

Note that $\nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t) > 0$ if $\nabla_{\vec{x}} f(\vec{x}_t) \neq \vec{0}$. Therefore, we can achieve arbitrarily low values by increasing $\alpha$.

However, it is very unlikely that our first-order Taylor approximation is accurate if we are far away from $\vec{x}_t$. Instead, we should stay *reasonably close* to $\vec{x}_t$ to ensure that our approximation is still valid. One way to ensure closeness is to punish the squared distance between $\vec{x}_{t+1}$ and $\vec{x}_t$, i.e. $\|\vec{x}_{t+1} - \vec{x}_t\|^2$. Accordingly, we obtain the alternative optimization problem

$$\min_{\vec{x}_{t+1} \in \mathbb{R}^K} \quad f(\vec{x}_t) + (\vec{x}_{t+1} - \vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t) + \frac{1}{2\alpha} \cdot \|\vec{x}_{t+1} - \vec{x}_t\|^2$$

where $\alpha > 0$ is a constant that regulates how much we wish to punish going away from $\vec{x}_t$. If $\alpha$ is small we nudge our solution to stay very close to $\vec{x}_t$, and if $\alpha$ is large we permit $\vec{x}_{t+1}$ to stray far away. So $\alpha$ can also be viewed as the trust we put in our first-order Taylor approximation or how daring we are in our approximation.

Now, let us solve this problem analytically. We first compute the gradient

$$\nabla_{\vec{x}_{t+1}} f(\vec{x}_t) + (\vec{x}_{t+1} - \vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t) + \frac{1}{2\alpha} \cdot \|\vec{x}_{t+1} - \vec{x}_t\|^2 = \nabla_{\vec{x}} f(\vec{x}_t) + \frac{1}{\alpha} \cdot (\vec{x}_{t+1} - \vec{x}_t)$$

By setting the gradient to zero we obtain:

$$\nabla_{\vec{x}} f(\vec{x}_t) + \frac{1}{\alpha} \cdot (\vec{x}_{t+1} - \vec{x}_t) = \vec{0} \quad \iff \quad \vec{x}_{t+1} = \vec{x}_t - \alpha \cdot \nabla_{\vec{x}} f(\vec{x}_t)$$

Note that this is exactly our step formula for gradient descent as seen in Algorithm 1, line 4. Also note that $\alpha$ is exactly the step size.

Just to be sure, let us also check the Hessian:

$$\nabla^2_{\vec{x}_{t+1}} f(\vec{x}_t) + (\vec{x}_{t+1} - \vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t) + \frac{1}{2\alpha} \cdot \|\vec{x}_{t+1} - \vec{x}_t\|^2 = \frac{1}{\alpha} \cdot \boldsymbol{I}^K$$

So our Hessian is the $K \times K$-dimensional identity matrix, multiplied with $\frac{1}{\alpha}$. The eigenvalues of this matrix are all $\frac{1}{\alpha} > 0$. Because this is a constant, our problem is convex and our solution in Algorithm 1, line 4, is indeed a global minimum.

It is also possible to show that gradient descent is guaranteed to converge if we start off close enough to a local minimum such that the second-order Taylor approximation is accurate and if $\alpha$ is small enough. However, we will omit this proof here for reasons of space.

#### Stochastic Gradient Descent / Adam

In cases where the objective function and its gradient are very expensive to compute, it may be helpful to not compute the gradient on the entire objective function but only over *a subset of the data*. More precisely, assume we can write the objective function as a sum, i.e.

$$f(\vec{x}) = \sum_{l=1}^{L} f_l(\vec{x})$$

where $f_l$ are functions $f_l : \mathbb{R}^K \to \mathbb{R}$ for all $l \in \{1, \ldots, L\}$. Then, we can also write the gradient as a sum, i.e.

$$\nabla_{\vec{x}} f(\vec{x}) = \sum_{l=1}^{L} \nabla_{\vec{x}} f_l(\vec{x}).$$

Accordingly, we can perform gradient descent by choosing, in each iteration, an $l \in \{1, \ldots, L\}$ randomly and performing a gradient step of the form $\vec{x}_{t+1} \leftarrow \vec{x}_t - \alpha \cdot \nabla_{\vec{x}} f_l(\vec{x})$. This is then called *stochastic gradient descent*.

A typical application example for stochastic gradient descent is machine learning, especially deep learning, i.e. optimizing the parameters of large artificial neural networks. In these cases, the objective function is a sum over a large number of data points. Computing the gradient over the entire data set would be costly. Therefore, we choose only a single data point, compute the gradient for this data point, and adjust the parameters right away. As long as we choose the data uniformly at random and optimize long enough, it can be shown that the resulting local minimum is very likely the same as with regular gradient descent. However, even if this local minimum is found, stochastic gradient will not converge because a local minimum for the entire objective function is not necessarily a local minimum for every element of the sum. Therefore, our variable values will fluctuate around the local minimum.

To prevent such fluctuations and to make the approach more numerically robust, we can choose a small set of the data instead of a single data point. This is then called *mini-batch gradient descent*, as shown in Algorithm 2.

**Algorithm 2** Minibatch gradient descent for the objective function $f = \sum_{l=1}^{L} f_l$, a starting value $\vec{x}_0 \in \mathbb{R}^K$, a step size $\alpha > 0$, a gradient threshold $\epsilon > 0$, and a minibatch size $R \in \mathbb{N}$.

**function** MINIBATCH GRADIENT DESCENT(objective function $f : \mathbb{R}^K \to \mathbb{R}$ with $f(\vec{x}) = \sum_{l=1}^{L} f_l(\vec{x})$, starting value $\vec{x}_0 \in \mathbb{R}^K$, step size $\alpha > 0$, threshold $\epsilon > 0$, minibatch size $R \in \mathbb{N}$)
  $t \leftarrow 0$.
  **while** true **do**
    Select $\{l_1, \ldots, l_R\} \subset \{1, \ldots, L\}$ at random.
    $\vec{p}_{t+1} \leftarrow -\frac{1}{R} \sum_{r=1}^{R} \nabla_{\vec{x}} f_{l_r}(\vec{x}_t)$.
    **if** $\|\vec{p}_{t+1}\| < \epsilon$ **then**
      **return** $\vec{x}_t$.
    **end if**
    $\vec{x}_{t+1} \leftarrow \vec{x}_t + \alpha \cdot \vec{p}_{t+1}$.
    $t \leftarrow t + 1$.
  **end while**
**end function**

In recent years, more advanced versions of the basic minibatch gradient descent scheme have emerged to make deep learning more efficient. One of the most prominent variations is *adaptive moment estimation (Adam)* (Kingma and Ba 2015), which makes two key changes. First, it introduces a momentum term which ensures that our variables tend to move into a consistent direction. Second, it adapts the step size such that the absolute change for every parameter in each iteration is the same. The algorithm is shown in Algorithm 3.

### Optimizing the Step Size

A weak point in standard gradient descent is that the step size $\alpha$ is a hyper-parameter which we have to choose. Ideally, we would like to choose an optimal $\alpha$ automatically, i.e. given a search direction $\vec{p}_t$ we wish to solve the optimization problem

$$\min_{\alpha > 0} \quad f(\vec{x}_t + \alpha \cdot \vec{p}_t). \tag{2.1}$$

Since this is a one-dimensional optimization problem (optimization along a line), solving this problem is also called *line search*. Combining line search with gradient descent results in the *steepest descent* method (refer to Algorithm 4).

The key question is how to solve Problem 2.1 efficiently. If $f$ is convex, this is relatively easy: We first consider the gradient of our objective function with respect to $\alpha$:

$$\frac{\partial}{\partial \alpha} f(\vec{x}_t + \alpha \cdot \vec{p}_t) = \nabla_{x_t} f(\vec{x}_t + \alpha \cdot \vec{p}_t)^T \cdot \vec{p}_t$$

Then, we increase $\alpha$ until the gradient becomes positive and then perform a binary search until our gradient vanishes. This works because if our gradient is negative initially and becomes positive at some point, it must cross zero somewhere in between. Also refer to Algorithm 5.

**Algorithm 3** Adaptive moment estimation (Adam) for the objective function $f = \sum_{l=1}^{L} f_l$, a starting value $\vec{x}_0 \in \mathbb{R}^K$, a step size $\alpha > 0$, a gradient threshold $\epsilon > 0$, memory parameters $\beta_1, \beta_2 \in [0, 1)$, and a minibatch size $R \in \mathbb{N}$.

**function** ADAM(objective function $f : \mathbb{R}^K \to \mathbb{R}$ with $f(\vec{x}) = \sum_{l=1}^{L} f_l(\vec{x})$, starting value $\vec{x}_0 \in \mathbb{R}^K$, step size $\alpha > 0$, threshold $\epsilon > 0$, memory parameters $\beta_1, \beta_2 \in [0, 1)$, minibatch size $R \in \mathbb{N}$)
  $t \leftarrow 0$. $\vec{p}_t \leftarrow \vec{0}$. $\vec{\sigma}_t \leftarrow \vec{0}$.
  **while** true **do**
    Select $\{l_1, \ldots, l_R\} \subset \{1, \ldots, L\}$ at random.
    $\vec{g}_{t+1} \leftarrow \frac{1}{R} \sum_{r=1}^{R} \nabla_{\vec{x}} f_{l_r}(\vec{x}_t)$.
    **if** $\|\vec{g}_{t+1}\| < \epsilon$ **then**
      **return** $\vec{x}_t$.
    **end if**
    $\vec{p}_{t+1} \leftarrow \beta_1 \cdot \vec{p}_t + (1 - \beta_1) \cdot (-\vec{g}_{t+1})$.
    $\alpha_t \leftarrow \alpha \cdot \sqrt{1 - (\beta_2)^t}/(1 - (\beta_1)^t)$.
    **for** $k \in \{1, \ldots, K\}$ **do**
      $\sigma^2_{t+1,k} \leftarrow \beta_2 \cdot \sigma^2_{t,k} + (1 - \beta_2) \cdot g^2_{t+1,k}$.
      $x_{t+1,k} \leftarrow x_{t,k} + \frac{\alpha_t}{\sigma_{t+1,k}} \cdot p_{t+1,k}$.
    **end for**
    $t \leftarrow t + 1$.
  **end while**
**end function**

**Algorithm 4** Steepest descent for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, a starting value $\vec{x}_0 \in \mathbb{R}^K$, and a gradient threshold $\epsilon > 0$.

**function** STEEPEST DESCENT(objective function $f : \mathbb{R}^K \to \mathbb{R}$, starting value $\vec{x}_0 \in \mathbb{R}^K$, threshold $\epsilon > 0$)
  $t \leftarrow 0$.
  **while** $\|\nabla_{\vec{x}} f(\vec{x}_t)\| > \epsilon$ **do**
    $\vec{p}_{t+1} \leftarrow -\nabla_{\vec{x}} f(\vec{x}_t)/\|\nabla_{\vec{x}} f(\vec{x}_t)\|$.
    $\alpha_{t+1} \leftarrow \operatorname{argmin}_{\alpha > 0} f(\vec{x}_t + \alpha \cdot \vec{p}_{t+1})$.
    $\vec{x}_{t+1} \leftarrow \vec{x}_t + \alpha_{t+1} \cdot \vec{p}_{t+1}$.
    $t \leftarrow t + 1$.
  **end while**
  **return** $\vec{x}_t$.
**end function**

**Algorithm 5** A binary search algorithm to realize a line search for for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, a current value $\vec{x}_t \in \mathbb{R}^K$, a search direction $\vec{p}_t \in \mathbb{R}^K$, and a gradient threshold $\epsilon > 0$.

**function** BINARY LINE SEARCH(objective function $f : \mathbb{R}^K \to \mathbb{R}$, current point $\vec{x}_t \in \mathbb{R}^K$, search direction $\vec{p}_t \in \mathbb{R}^K$, threshold $\epsilon > 0$)
  $lo \leftarrow 0$.
  $hi \leftarrow 1$.
  **while** $\nabla_{x_t} f(\vec{x}_t + hi \cdot \vec{p}_t)^T \cdot \vec{p}_t < 0$ **do**
    $lo \leftarrow hi$.
    $hi \leftarrow hi \cdot 2$.
  **end while**
  $\alpha \leftarrow \frac{lo+hi}{2}$.
  **while** $\|\nabla_{x_t} f(\vec{x}_t + \alpha \cdot \vec{p}_t)^T \cdot \vec{p}_t\| > \epsilon$ **do**
    **if** $\nabla_{x_t} f(\vec{x}_t + \alpha \cdot \vec{p}_t)^T \cdot \vec{p}_t < 0$ **then**
      $lo \leftarrow \alpha$.
    **else**
      $hi \leftarrow \alpha$.
    **end if**
    $\alpha \leftarrow \frac{lo+hi}{2}$.
  **end while**
  **return** $\alpha$.
**end function**

In practice, however, such a binary search scheme is not used for multiple reasons. First, because convexity is not always given, and second, because it still requires a lot of gradient evaluations which may be expensive. Therefore, most approaches settle for a step size that is "good enough", where "good enough" is defined by the so-called Wolfe conditions.

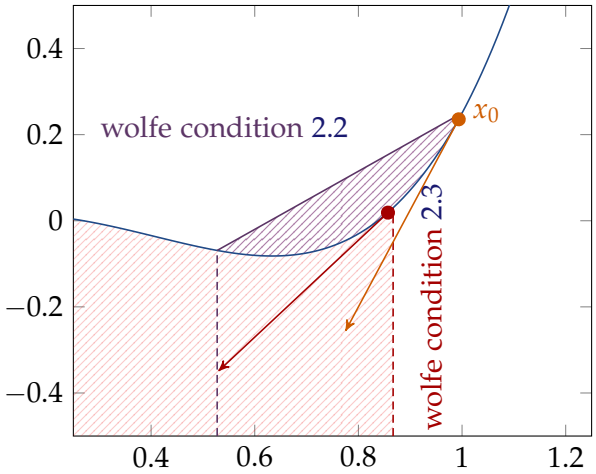



*Figure 2.2:* An illustration of the $(0.3, 0.5)$-Wolfe conditions when going from $x_0 = 1$ into direction $\vec{p} = -\nabla_x f(1) = -2.25$ on $f(x) = x^4 - x^2 + \frac{1}{4}x$. The purple striped region indicates where the first Wolfe condition 2.2 is fulfilled and the red dashed region indicates where the second Wolfe condition 2.3 is fulfilled.

**Definition 71** (Wolfe conditions)**.** Let $f$ be a differentiable function $f : \mathbb{R}^K \to \mathbb{R}$, let $\vec{x}_t \in \mathbb{R}^K$ be some vector, let $\vec{p} \in \mathbb{R}^K$ be another vector with $\vec{p}^T \cdot \nabla_{\vec{x}} f(\vec{x}_t) < 0$, which we call *direction*, let $c_1, c_2 \in \mathbb{R}$ be some constants with $0 < c_1 < c_2 < 1$, and let $\alpha \in \mathbb{R}$ with $\alpha > 0$. We say that $\alpha$ conforms to the $(c_1, c_2)$-Wolfe conditions when going from $\vec{x}_t$ into direction $\vec{p}$ on $f$ if and only if:

$$f(\vec{x}_t + \alpha \cdot \vec{p}) \leqslant f(\vec{x}_t) + c_1 \cdot \alpha \cdot \vec{p}^T \cdot \nabla_{\vec{x}} f(\vec{x}_t) \tag{2.2}$$

$$\nabla_{\vec{x}} f(\vec{x}_t + \alpha \cdot \vec{p})^T \cdot \vec{p} \geqslant c_2 \cdot \nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \vec{p} \tag{2.3}$$

**Remark 72** (Interpretation of the Wolfe conditions)**.** The first Wolfe condition 2.2 compares the actual objective function value at position $\vec{x}_t + \alpha \cdot \vec{p}$ with the value of the first-order Taylor approximation, where the slope of the approximation is multiplied by $c_1$. In other words, our first Wolfe condition is that our objective function value should decrease at least as much as the first-order Taylor approximation would predict, relaxed by a factor $c_1$. This relaxation is necessary because, for (locally) convex functions, the actual objective function value is always higher than the value predicted by the first-order Taylor approximation (refer to Theorem 41), such that we would never be able to fulfill this criterion without relaxation.

The second Wolfe condition 2.3 is related to the gradient of $f$ *with respect to* $\alpha$. In particular, we can define the function $g : \mathbb{R} \to \mathbb{R}$ with $g(\alpha) := f(\vec{x}_t + \alpha \cdot \vec{p})$ with the derivative

$$\frac{\partial}{\partial \alpha} g(\alpha) = \nabla_{\vec{x}} f(\vec{x}_t + \alpha \cdot \vec{p})^T \cdot \frac{\partial}{\partial \alpha} \cdot (\vec{x}_t + \alpha \cdot \vec{p}) = \nabla_{\vec{x}} f(\vec{x}_t + \alpha \cdot \vec{p})^T \cdot \vec{p}$$

This yields the special case $\frac{\partial}{\partial \alpha} g(0) = \nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \vec{p}$. Accordingly, the second Wolfe condition 2.3 requires that our gradient should shrink by a factor of at least $c_2$. Note that we use $\geqslant$ because the right-hand side of condition 2.3 is guaranteed to be negative due to $\vec{p}^T \cdot \nabla_{\vec{x}} f(\vec{x}_t) < 0$ and $c_2 > 0$.

The Wolfe conditions are illustrated in Figure 2.2 for an example function.

### Conjugate Gradient

A problem remaining with steepest descent is its convergence speed. If we are unlucky, subsequent gradient directions may be highly correlated and only very small steps occur. The idea of the *conjugate gradient* method is to use search directions which are, in some sense, orthogonal to each other such that we discover directions in which improvement is still possible.

The exact derivation of the conjugate gradient method is based on solving linear equation systems, which confuses more than it helps. For the purpose of this document, we simply define two direction vectors $\vec{p}_{t+1} \in \mathbb{R}^K$ and $\vec{p}_t \in \mathbb{R}^K$ as *conjugate* with respect to function $f : \mathbb{R}^K \to \mathbb{R}$ and position $\vec{x}_t$ if it holds:

$$\vec{p}_{t+1}^T \cdot \nabla_{\vec{x}}^2 f(\vec{x}_t) \cdot \vec{p}_t = 0. \tag{2.4}$$

An unfortunate property of Definition 2.4 is that it relies on the Hessian of $f$, which is quadratic in $K$. To approximate Equation 2.4, we can instead define the search direction in step $t+1$ as $\vec{p}_{t+1} := -\nabla_{\vec{x}} f(\vec{x}_t) + \beta_{t+1} \cdot \vec{p}_t$ with a *correction factor* $\beta_{t+1}$ that is computed via one of the following equations.

$$\beta_{t+1} = \frac{\nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t)}{\nabla_{\vec{x}} f(\vec{x}_{t-1})^T \cdot \nabla_{\vec{x}} f(\vec{x}_{t-1})} \qquad \text{(Fletcher-Reeves)}$$

$$\beta_{t+1} = \frac{\nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \left(\nabla_{\vec{x}} f(\vec{x}_t) - \nabla_{\vec{x}} f(\vec{x}_{t-1})\right)}{\nabla_{\vec{x}} f(\vec{x}_{t-1})^T \cdot \nabla_{\vec{x}} f(\vec{x}_{t-1})} \qquad \text{(Polak-Ribière)}$$

$$\beta_{t+1} = \frac{\nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \left(\nabla_{\vec{x}} f(\vec{x}_t) - \nabla_{\vec{x}} f(\vec{x}_{t-1})\right)}{\vec{p}_t^T \cdot \left(\nabla_{\vec{x}} f(\vec{x}_t) - \nabla_{\vec{x}} f(\vec{x}_{t-1})\right)} \qquad \text{(Hestenes-Stiefel)}$$

The resulting conjugate gradient algorithm is shown in Algorithm 6. Figure 2.3 illustrates how gradient descent, steepest descent, and conjugate gradient compare in minimizing a benchmark objective function.

**Algorithm 6** The conjugate gradient method for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, a starting value $\vec{x}_0 \in \mathbb{R}^K$, and a gradient threshold $\epsilon > 0$.

**function** CONJUGATE GRADIENT(objective function $f : \mathbb{R}^K \to \mathbb{R}$, starting value $\vec{x}_0 \in \mathbb{R}^K$, threshold $\epsilon > 0$)
  $\vec{p}_1 \leftarrow -\nabla_{\vec{x}} f(\vec{x}_0)$.
  $\alpha_1 \leftarrow \operatorname{argmin}_{\alpha>0} f(\vec{x}_0 + \alpha \cdot \vec{p}_1)$.
  $\vec{x}_1 \leftarrow \vec{x}_0 + \alpha_1 \cdot \vec{p}_1$.
  $t \leftarrow 1$.
  **while** $\|\nabla_{\vec{x}} f(\vec{x}_t)\| > \epsilon$ **do**
    Compute $\beta_{t+1}$ according to Fletcher-Reeves, Polak-Ribière, or Hestenes-Stiefel.
    $\vec{p}_{t+1} \leftarrow -\nabla_{\vec{x}} f(\vec{x}_t) + \beta_{t+1} \cdot \vec{p}_t$.
    $\alpha_{t+1} \leftarrow \operatorname{argmin}_{\alpha>0} f(\vec{x}_t + \alpha \cdot \vec{p}_{t+1})$.
    $\vec{x}_{t+1} \leftarrow \vec{x}_t + \alpha_{t+1} \cdot \vec{p}_{t+1}$.
    $t \leftarrow t + 1$.
  **end while**
  **return** $\vec{x}_t$.
**end function**

### Newton's Method

The previous methods have in common that they rely on the first-order Taylor approximation, which is also why they are called *first-order* methods. However, we could also try to use the *second-order* Taylor approximation as basis for our optimization algorithm. Indeed, repeating the same derivation as for gradient descent with the second-order Taylor approximation yields *Newton's method*, as we will see now.

Assume we wish to minimize the objective function $f : \mathbb{R}^K \to \mathbb{R}$, using the second-order Taylor approximation around point $\vec{x}_t \in \mathbb{R}^K$. As before, we need to ensure that we stay close to $\vec{x}_t$ to ensure that our approximation remains valid[1]. Therefore, we consider the optimization problem

$$\min_{\vec{x}_{t+1} \in \mathbb{R}^k} \quad \tilde{f}^2_{\vec{x}_t}(\vec{x}_{t+1}) + \frac{1}{2\alpha}\|\vec{x}_{t+1} - \vec{x}_t\|^2$$

where $\tilde{f}^2_{\vec{x}_t}(\vec{x}_{t+1})$ is given as in Equation 1.26, i.e.

$$\min_{\vec{x}_{t+1} \in \mathbb{R}^k} \quad f(\vec{x}_t) + (\vec{x}_{t+1} - \vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t) + \frac{1}{2}(\vec{x}_{t+1} - \vec{x}_t)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot (\vec{x}_{t+1} - \vec{x}_t) + \frac{1}{2\alpha}\|\vec{x}_{t+1} - \vec{x}_t\|^2 \tag{2.5}$$

1 Note that this argument is usually not given in derivations for Newton's method. The "classic" version of Newton's method can be recovered by setting $\alpha$ to a very large value.

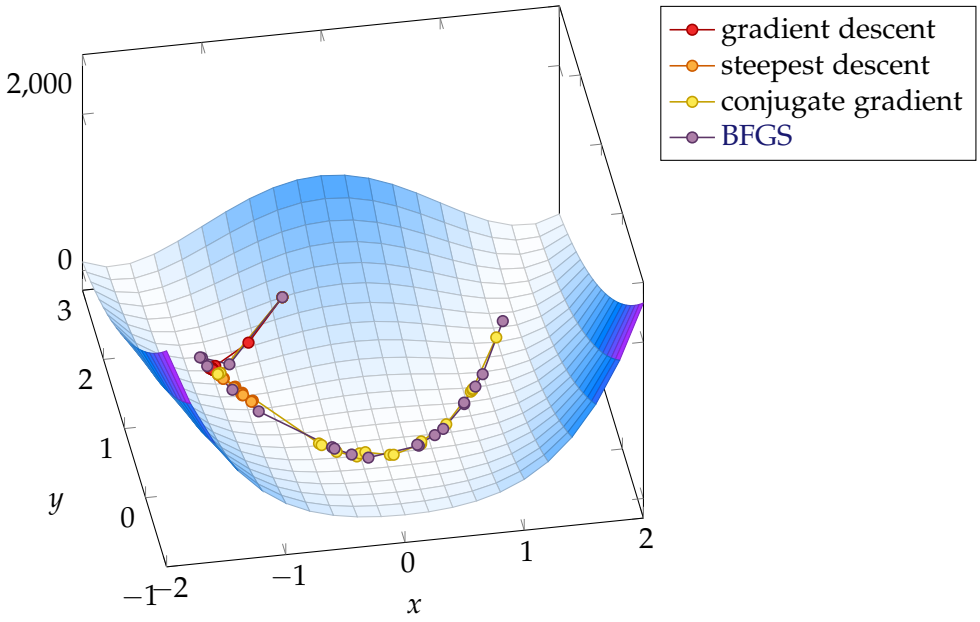



*Figure 2.3:* A comparison of 20 iterations of gradient descent, steepest descent, conjugate gradient, and BFGS on Rosenbrock's function $f(x,y) = (\frac{3}{2} - x)^2 + 100 \cdot (y - x^2)^2$, starting from $\vec{x}_0 = (-\frac{1}{2}, 2)$ and using the binary line search algorithm 5. Gradient descent achieves an objective value $f(x^*, y^*) \approx 7.22$, steepest descent achieves $f(x^*, y^*) \approx 6.01$, conjugate gradient achieves $f(x^*, y^*) \approx 0.17$, and BFGS achieves $f(x^*, y^*) \approx 0.06$.

To solve this problem, we first consider the gradient.

$$\nabla_{\vec{x}_{t+1}} \tilde{f}^2_{\vec{x}_t}(\vec{x}_{t+1}) + \frac{1}{2\alpha}\|\vec{x}_{t+1} - \vec{x}_t\|^2 = \nabla_{\vec{x}} f(\vec{x}_t) + \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot (\vec{x}_{t+1} - \vec{x}_t) + \frac{1}{\alpha} \cdot (\vec{x}_{t+1} - \vec{x}_t)$$

By setting the gradient to zero we obtain:

$$\begin{aligned}
\nabla_{\vec{x}} f(\vec{x}_t) + \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot (\vec{x}^*_{t+1} - \vec{x}_t) + \frac{1}{\alpha} \cdot (\vec{x}^*_{t+1} - \vec{x}_t) &= 0 \\
\left(\nabla^2_{\vec{x}} f(\vec{x}_t) + \frac{1}{\alpha} \cdot \boldsymbol{I}^K\right) \cdot (\vec{x}^*_{t+1} - \vec{x}_t) &= -\nabla_{\vec{x}} f(\vec{x}_t) \\
\vec{x}^*_{t+1} &= \vec{x}_t - \left(\nabla^2_{\vec{x}} f(\vec{x}_t) + \frac{1}{\alpha} \cdot \boldsymbol{I}^K\right)^{-1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)
\end{aligned}$$

Now, consider the Hessian of our problem.

$$\nabla^2_{\vec{x}_{t+1}} \tilde{f}^2_{\vec{x}_t}(\vec{x}_{t+1}) + \frac{1}{2\alpha}\|\vec{x}_{t+1} - \vec{x}_t\|^2 = \nabla^2_{\vec{x}} f(\vec{x}_t) + \frac{1}{\alpha} \cdot \boldsymbol{I}^K$$

Note that the matrix $\frac{1}{\alpha} \cdot \boldsymbol{I}^K$ is definitely positive definite because all eigenvalues are $\frac{1}{\alpha} > 0$. The eigenvalues of our matrix sum are exactly the eigenvalues of $\nabla^2_{\vec{x}} f(\vec{x}_t)$ plus the term $\frac{1}{\alpha}$. Therefore, we obtain a local minimum of the second-order Taylor approximation if all eigenvalues of the objective function Hessian $\nabla^2_{\vec{x}} f(\vec{x}_t)$ are above $-\frac{1}{\alpha}$.

Also note that we can ensure convexity of the second-order Taylor approximation by setting $\alpha$ small enough, because this will ensure overall positive eigenvalues. So for small enough $\alpha$, $\vec{x}^*_{t+1}$ is indeed a global minimum of the second-order Taylor approximation.

Beyond the issues of convexity and numerical stability, though, we can set $\alpha$ pretty much arbitrary, which means that we do not require a step width optimization for Newton's method.

The final version of Newton's method is shown in Algorithm 7.

**Algorithm 7** Newton's method for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, a starting value $\vec{x}_0 \in \mathbb{R}^K$, a regularization constant $\alpha > 0$, and a gradient threshold $\epsilon > 0$.

**function** NEWTON(objective function $f : \mathbb{R}^K \to \mathbb{R}$, starting value $\vec{x}_0 \in \mathbb{R}^K$, regularization $\alpha > 0$, threshold $\epsilon > 0$)
  $t \leftarrow 0$.
  **while** $\|\nabla_{\vec{x}} f(\vec{x}_t)\| > \epsilon$ **do**
    $\vec{x}_{t+1} \leftarrow \vec{x}_t - \left(\nabla^2_{\vec{x}} f(\vec{x}_t) + \frac{1}{\alpha} \cdot \boldsymbol{I}^K\right)^{-1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)$.
    $t \leftarrow t + 1$.
  **end while**
  **return** $\vec{x}_t$.
**end function**

### (L-)BFGS

A key advantage compared to first-order methods is that Newton's method typically requires far less iterations to arrive at a local minimum because the second-order Taylor approximation captures the local behavior better. Unfortunately, though, the Newton's method requires us to compute the inverse of the Hessian matrix which takes $\mathcal{O}(K^3)$ operations, such that the method becomes practically unusable if we have many variables. To alleviate this problem and still exploit second-order information, so-called *quasi-Newton* methods have emerged which approximate the inverse of the Hessian using gradient information, the best-known of which is the Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS).

The derivation for BFGS is, unfortunately, quite involved. We do show it here in full for interested readers, though. The derivation has two parts. First, we obtain a recursive update formula for an approximation of the Hessian. Second, we use this update formula and the matrix inversion Lemma to achieve instead an update formula for the *inverse* of the Hessian.

Our starting point is that we wish to find a matrix $\boldsymbol{H}_t$ at iteration $t$ which is not the Hessian but is "Hessian-like". One way to formalize "Hessian-likeness" is the secant-equation which states that the Hessian is precisely the matrix that maps differences between the inputs of a function to differences between the gradient at those points, i.e.:

$$\boldsymbol{H}_t \cdot (\vec{x}_t - \vec{x}_{t-1}) = \nabla_{\vec{x}} f(\vec{x}_t) - \nabla_{\vec{x}} f(\vec{x}_{t-1}) \tag{2.6}$$

To make our notation a bit less cluttered, let's define $\vec{\Delta}_t := \vec{x}_t - \vec{x}_{t-1}$ and $\vec{\delta}_t := \nabla_{\vec{x}} f(\vec{x}_t) - \nabla_{\vec{x}} f(\vec{x}_{t-1})$, such that we can re-write Equation 2.6 as $\boldsymbol{H}_t \cdot \vec{\Delta}_t = \vec{\delta}_t$.

Next, we apply a clever trick impose a pre-defined form on our update formula. In particular, we would like to obtain an update of the following shape.

$$\boldsymbol{H}_t = \boldsymbol{H}_{t-1} + \alpha \cdot \vec{\delta}_t \cdot \vec{\delta}_t^T + \beta \cdot \boldsymbol{H}_{t-1} \cdot \vec{\Delta}_t \cdot \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \tag{2.7}$$

for some numbers $\alpha, \beta \in \mathbb{R}$. Now, we can plug in our update formula 2.7 into Equation 2.6.

$$\left(\boldsymbol{H}_{t-1} + \alpha \cdot \vec{\delta}_t \cdot \vec{\delta}_t^T + \beta \cdot \boldsymbol{H}_{t-1} \cdot \vec{\Delta}_t \cdot \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T\right) \cdot \vec{\Delta}_t = \vec{\delta}_t$$

$$\iff \quad \vec{\delta}_t \cdot \left(\alpha \cdot \vec{\delta}_t^T \cdot \vec{\Delta}_t\right) + \boldsymbol{H}_{t-1} \cdot \vec{\Delta}_t \cdot \left(1 + \beta \cdot \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \cdot \vec{\Delta}_t\right) = \vec{\delta}_t$$

By inspecting this equation we see that one solution could be obtained if $\alpha \cdot \vec{\delta}_t^T \cdot \vec{\Delta}_t = 1$ and $1 + \beta \cdot \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \cdot \vec{\Delta}_t = 0$. We can use these two equations to solve for $\alpha$ and $\beta$:

$$\begin{aligned} \alpha \cdot \vec{\delta}_t^T \cdot \vec{\Delta}_t = 1 \qquad &\iff \quad \alpha = \frac{1}{\vec{\delta}_t^T \cdot \vec{\Delta}_t} \\ 1 + \beta \cdot \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \cdot \vec{\Delta}_t = 0 \qquad &\iff \quad \beta = -\frac{1}{\vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \cdot \vec{\Delta}_t} \end{aligned}$$

By plugging these results into Equation 2.7 we obtain:

$$\boldsymbol{H}_t = \boldsymbol{H}_{t-1} + \frac{\vec{\delta}_t \cdot \vec{\delta}_t^T}{\vec{\delta}_t^T \cdot \vec{\Delta}_t} - \frac{\boldsymbol{H}_{t-1} \cdot \vec{\Delta}_t \cdot \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T}{\vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \cdot \vec{\Delta}_t}$$

So we now have an iterative update formula for an approximation of the Hessian using only gradient information. So far, so good. However, this does not yet solve our time efficiency problem because we still would need to invert $\boldsymbol{H}_t$ in each time step, which would take cubic time. Fortunately, though, because we have our update formula, we can express the inverse of $\boldsymbol{H}_t$ in a more efficient way using the Woodbury matrix identity, also called the matrix inversion lemma.

In particular, the Woodbury matrix identity states that for any four matrices $\boldsymbol{A} \in \mathbb{R}^{K \times K}$, $\boldsymbol{U} \in \mathbb{R}^{K \times n}$, $\boldsymbol{C} \in \mathbb{R}^{n \times n}$, and $\boldsymbol{V} \in \mathbb{R}^{n \times K}$ it holds:

$$(\boldsymbol{A} + \boldsymbol{U} \cdot \boldsymbol{C} \cdot \boldsymbol{V})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1} \cdot \boldsymbol{U} \cdot \left(\boldsymbol{C}^{-1} + \boldsymbol{V} \cdot \boldsymbol{A}^{-1} \cdot \boldsymbol{U}\right)^{-1} \cdot \boldsymbol{V} \cdot \boldsymbol{A}^{-1}$$

We can re-write the right-hand-side of our update formula 2.7 in terms of the left-hand side of the Woodbury matrix identity by setting:

$$\boldsymbol{A} = \boldsymbol{H}_{t-1}, \quad \boldsymbol{U} = \begin{pmatrix} \boldsymbol{H}_{t-1} \cdot \vec{\Delta}_t & \vec{\delta}_t \end{pmatrix}, \quad \boldsymbol{C} = \begin{pmatrix} \beta & 0 \\ 0 & \alpha \end{pmatrix}, \text{ and } \quad \boldsymbol{V} = \begin{pmatrix} \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \\ \vec{\delta}_t \end{pmatrix},$$

which yields:

$$\begin{aligned} \boldsymbol{A} + \boldsymbol{U} \cdot \boldsymbol{C} \cdot \boldsymbol{V} &= \boldsymbol{H}_{t-1} + \begin{pmatrix} \boldsymbol{H}_{t-1} \cdot \vec{\Delta}_t & \vec{\delta}_t \end{pmatrix} \cdot \begin{pmatrix} \beta & 0 \\ 0 & \alpha \end{pmatrix} \cdot \begin{pmatrix} \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \\ \vec{\delta}_t^T \end{pmatrix} \\ &= \boldsymbol{H}_{t-1} + \boldsymbol{H}_{t-1} \cdot \vec{\Delta}_t \cdot \beta \cdot \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T + \vec{\delta}_t \cdot \alpha \cdot \vec{\delta}_t^T = \boldsymbol{H}_t \end{aligned}$$

as desired.

Plugging this into the Woodbury matrix identity we obtain:

$$\begin{aligned} \boldsymbol{H}_t^{-1} =& \boldsymbol{H}_{t-1}^{-1} - \boldsymbol{H}_{t-1}^{-1} \cdot \begin{pmatrix} \boldsymbol{H}_{t-1} \cdot \vec{\Delta}_t & \vec{\delta}_t \end{pmatrix} \\ & \cdot \left( \begin{pmatrix} \beta & 0 \\ 0 & \alpha \end{pmatrix}^{-1} + \begin{pmatrix} \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \\ \vec{\delta}_t^T \end{pmatrix} \cdot \boldsymbol{H}_{t-1}^{-1} \cdot \begin{pmatrix} \boldsymbol{H}_{t-1} \cdot \vec{\Delta}_t & \vec{\delta}_t \end{pmatrix} \right)^{-1} \cdot \begin{pmatrix} \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \\ \vec{\delta}_t^T \end{pmatrix} \cdot \boldsymbol{H}_{t-1}^{-1} \\ =& \boldsymbol{H}_{t-1}^{-1} - \begin{pmatrix} \vec{\Delta}_t & \boldsymbol{H}_{t-1}^{-1} \cdot \vec{\delta}_t \end{pmatrix} \\ & \cdot \left( \begin{pmatrix} \frac{1}{\beta} & 0 \\ 0 & \frac{1}{\alpha} \end{pmatrix} + \begin{pmatrix} \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \\ \vec{\delta}_t^T \end{pmatrix} \cdot \begin{pmatrix} \vec{\Delta}_t & \boldsymbol{H}_{t-1}^{-1} \cdot \vec{\delta}_t \end{pmatrix} \right)^{-1} \cdot \begin{pmatrix} \vec{\Delta}_t^T \\ \vec{\delta}_t^T \cdot \boldsymbol{H}_{t-1}^{-1} \end{pmatrix} \end{aligned}$$

$$= \boldsymbol{H}_{t-1}^{-1} - \begin{pmatrix} \vec{\Delta}_t & \boldsymbol{H}_{t-1}^{-1} \cdot \vec{\delta}_t \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{\beta} + \vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \cdot \vec{\Delta}_t & \vec{\Delta}_t^T \cdot \vec{\delta}_t \\ \vec{\delta}_t^T \cdot \vec{\Delta}_t & \frac{1}{\alpha} + \vec{\delta}_t^T \cdot \boldsymbol{H}_{t-1}^{-1} \cdot \vec{\delta}_t \end{pmatrix}^{-1} \cdot \begin{pmatrix} \vec{\Delta}_t^T \\ \vec{\delta}_t^T \cdot \boldsymbol{H}_{t-1}^{-1} \end{pmatrix}$$

Now, recall that $\frac{1}{\beta} = -\vec{\Delta}_t^T \cdot {\boldsymbol{H}_{t-1}}^T \cdot \vec{\Delta}_t$ and $\frac{1}{\alpha} = \vec{\delta}_t^T \cdot \vec{\Delta}_t$. Accordingly, we can re-write our expression as follows.

$$\begin{aligned}
\boldsymbol{H}_t^{-1} =& \boldsymbol{H}_{t-1}^{-1} - \begin{pmatrix} \vec{\Delta}_t & \boldsymbol{H}_{t-1}^{-1} \cdot \vec{\delta}_t \end{pmatrix} \cdot \begin{pmatrix} 0 & \frac{1}{\alpha} \\ \frac{1}{\alpha} & \frac{1}{\alpha} + \vec{\delta}_t^T \cdot \boldsymbol{H}_{t-1}^{-1} \cdot \vec{\delta}_t \end{pmatrix}^{-1} \cdot \begin{pmatrix} \vec{\Delta}_t^T \\ \vec{\delta}_t^T \cdot \boldsymbol{H}_{t-1}^{-1} \end{pmatrix} \\
=& \boldsymbol{H}_{t-1}^{-1} - \begin{pmatrix} \vec{\Delta}_t & \boldsymbol{H}_{t-1}^{-1} \cdot \vec{\delta}_t \end{pmatrix} \cdot \begin{pmatrix} -\alpha^2 \cdot (\frac{1}{\alpha} + \vec{\delta}_t^T \cdot \boldsymbol{H}_{t-1}^{-1} \cdot \vec{\delta}_t) & \alpha \\ \alpha & 0 \end{pmatrix} \cdot \begin{pmatrix} \vec{\Delta}_t^T \\ \vec{\delta}_t^T \cdot \boldsymbol{H}_{t-1}^{-1} \end{pmatrix} \\
=& \boldsymbol{H}_{t-1}^{-1} - \begin{pmatrix} \vec{\Delta}_t & \boldsymbol{H}_{t-1}^{-1} \cdot \vec{\delta}_t \end{pmatrix} \cdot \begin{pmatrix} -\alpha^2 \cdot (\frac{1}{\alpha} + \vec{\delta}_t^T \cdot \boldsymbol{H}_{t-1}^{-1} \cdot \vec{\delta}_t) \cdot \vec{\Delta}_t^T + \alpha \cdot \vec{\delta}_t^T \cdot \boldsymbol{H}_{t-1}^{-1} \\ \alpha \cdot \vec{\Delta}_t^T \end{pmatrix} \\
=& \boldsymbol{H}_{t-1}^{-1} + \alpha^2 \cdot \vec{\Delta}_t \cdot \left(\frac{1}{\alpha} + \vec{\delta}_t^T \cdot \boldsymbol{H}_{t-1}^{-1} \cdot \vec{\delta}_t\right) \cdot \vec{\Delta}_t^T - \alpha \cdot \vec{\Delta}_t \cdot \vec{\delta}_t^T \cdot \boldsymbol{H}_{t-1}^{-1} - \alpha \cdot \boldsymbol{H}_{t-1}^{-1} \cdot \vec{\delta}_t \cdot \vec{\Delta}_t^T \\
=& (\boldsymbol{I}^K - \alpha \cdot \vec{\Delta}_t \cdot \vec{\delta}_t^T) \cdot \boldsymbol{H}_{t-1}^{-1} \cdot (\boldsymbol{I}^K - \alpha \cdot \vec{\delta}_t \cdot \vec{\Delta}_t^T) + \alpha \cdot \vec{\Delta}_t \cdot \vec{\Delta}_t^T
\end{aligned}$$

By plugging in our expression for $\alpha$ we obtain the final BFGS update formula.

$$\boldsymbol{H}_t^{-1} = \left(\boldsymbol{I}^K - \frac{\vec{\Delta}_t \cdot \vec{\delta}_t^T}{\vec{\Delta}_t^T \cdot \vec{\delta}_t}\right) \cdot \boldsymbol{H}_{t-1}^{-1} \cdot \left(\boldsymbol{I}^K - \frac{\vec{\delta}_t \cdot \vec{\Delta}_t^T}{\vec{\delta}_t^T \cdot \vec{\Delta}_t}\right) + \frac{\vec{\Delta}_t \cdot \vec{\Delta}_t^T}{\vec{\Delta}_t^T \cdot \vec{\delta}_t} \tag{2.8}$$

The resulting BFGS algorithm is shown in Algorithm 8.

**Algorithm 8** The Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS) for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, a starting value $\vec{x}_0 \in \mathbb{R}^K$, and a gradient threshold $\epsilon > 0$.

1: **function** BFGS(objective function $f : \mathbb{R}^K \to \mathbb{R}$, starting value $\vec{x}_0 \in \mathbb{R}^K$, threshold $\epsilon > 0$)
2: $t \leftarrow 0$.
3: $\boldsymbol{H}_0^{-1} \leftarrow$ Equation 2.8 with $\boldsymbol{H}_{t-1}^{-1} = \boldsymbol{I}^K$.
4: **while** $\|\nabla_{\vec{x}} f(\vec{x}_t)\| > \epsilon$ **do**
5: $\vec{p}_{t+1} \leftarrow -\boldsymbol{H}_t^{-1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)$.
6: $\alpha_{t+1} \leftarrow \text{argmin}_{\alpha > 0} f(\vec{x}_t + \alpha \cdot \vec{p}_{t+1})$.
7: $\vec{x}_{t+1} \leftarrow \vec{x}_t + \alpha_{t+1} \cdot \vec{p}_{t+1}$.
8: $t \leftarrow t + 1$.
9: $\boldsymbol{H}_t^{-1} \leftarrow$ Equation 2.8.
10: **end while**
11: **return** $\vec{x}_t$.
12: **end function**

Note that our BFGS update formula still has quadratic complexity due to the multiplications involved. We achieve a linear-time (and linear-memory) version by observing that we do not even need the inverted Hessian as such, but only the product $\boldsymbol{H}_t^{-1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)$. This product can be computed based on more efficient formulae, which then yield the so-called *limited-memory* BFGS algorithm (L-BFGS Nocedal and Wright 1999).

### Trust Region Method

In all our methods up until now we prevented a too large distance between our current point $\vec{x}_t$ and our next point $\vec{x}_{t+1}$ by adding a punishment term to our objective function.

Alternatively, we could also decide to introduce a fixed constraint which ensures that we stay in a pre-defined radius – the so-called *trust region*. More specifically, for a twice differentiable objective function $f : \mathbb{R}^K \to \mathbb{R}$, a current point $\vec{x}_t$, and a radius $\Delta$ we obtain the following optimization problem.

$$\begin{aligned} \min_{\vec{x}_{t+1} \in \mathbb{R}^K} \quad & f(\vec{x}_t) + (\vec{x}_{t+1} - \vec{x}_t) \cdot \nabla_{\vec{x}} f(\vec{x}_t) + \frac{1}{2} (\vec{x}_{t+1} - \vec{x}_t)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot (\vec{x}_{t+1} - \vec{x}_t) \\ \text{s.t.} \quad & \|\vec{x}_{t+1} - \vec{x}_t\| \leqslant \Delta \end{aligned} \tag{2.9}$$

Note the similarity to the Newton-method problem in Equation 2.5.

If we try to solve this problem analytically, we obtain the following Lagrange dual.

$$\begin{aligned} \sup_{\lambda \in \mathbb{R}} \quad \inf_{\vec{x}_{t+1} \in \mathbb{R}^K} \quad & f(\vec{x}_t) + (\vec{x}_{t+1} - \vec{x}_t) \cdot \nabla_{\vec{x}} f(\vec{x}_t) + \frac{1}{2} (\vec{x}_{t+1} - \vec{x}_t)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot (\vec{x}_{t+1} - \vec{x}_t) \\ & - \lambda \cdot \left( \Delta^2 - \|\vec{x}_{t+1} - \vec{x}_t\|^2 \right) \\ \text{s.t.} \quad & \lambda \geqslant 0 \end{aligned}$$

By comparing this problem to Equation 2.5, it becomes obvious that our solution will be:

$$\vec{x}^*_{t+1} = \vec{x}_t - \left( \nabla^2_{\vec{x}} f(\vec{x}_t) + 2\lambda \cdot \boldsymbol{I}^K \right)^{-1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)$$

subject to the KKT:

$$\lambda \geqslant 0, \quad \|\vec{x}_{t+1} - \vec{x}_t\|^2 \leqslant \Delta^2, \text{ and } \quad \lambda \cdot \left( \Delta^2 - \|\vec{x}_{t+1} - \vec{x}_t\|^2 \right) = 0.$$

Note that we do not know which value $\lambda$ takes, precisely. Either the classic Newton step of $\vec{x}^*_{t+1} = \vec{x}_t - \left( \nabla^2_{\vec{x}} f(\vec{x}_t) \right)^{-1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)$ for $\lambda = 0$ is feasible, or we have $\lambda > 0$ such that

$$\|\vec{x}_{t+1} - \vec{x}_t\|^2 = \Delta^2 \quad \iff \quad \|\left( \nabla^2_{\vec{x}} f(\vec{x}_t) + 2\lambda \cdot \boldsymbol{I}^K \right)^{-1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)\|^2 = \Delta^2,$$

This is a global minimum if the Hessian of the Lagrangian $\nabla^2_{\vec{x}} f(\vec{x}_t) + 2\lambda \cdot \boldsymbol{I}^K$ is positive definite.

An unfortunate drawback of this solution is that the equation $\|\left( \nabla^2_{\vec{x}} f(\vec{x}_t) + 2\lambda \cdot \boldsymbol{I}^K \right)^{-1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)\|^2 = \Delta^2$ has no analytical solution for $\lambda$. Neither can we optimize $\lambda$ efficiently because for any example value of $\lambda$ we would have to perform a matrix inversion.

So, what can we do instead? The simplest possible approach is to take the classic Newton step as direction vector, i.e. $\vec{p}_{t+1} = -\left( \nabla^2_{\vec{x}} f(\vec{x}_t) \right)^{-1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)$, and to set the step size $\alpha_{t+1}$ as either 1 if $\|\vec{p}_{t+1}\| \leqslant \Delta$ or as $\alpha_{t+1} = \Delta / \|\vec{p}_{t+1}\|$ otherwise, which overall results in $\alpha_{t+1} = \min\{1, \Delta / \|\vec{p}_{t+1}\|\}$.

Note that we can combine this scheme with approximation schemes for the Hessian, such as BFGS.

An alternative route would be to not use the Newton step as direction vector but instead the negative gradient, which results in the so-called *Cauchy point*. In more detail, the Cauchy point is the optimal next point if we take the negative gradient as direction vector and optimize the step size to minimize the second-order Taylor approximation, i.e.:

$$\begin{aligned} \min_{\alpha_{t+1} \in \mathbb{R}} \quad & f(\vec{x}_t) - \alpha_{t+1} \nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t) + \frac{1}{2} \alpha^2_{t+1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot \nabla_{\vec{x}} f(\vec{x}_t) \\ \text{s.t.} \quad & \alpha_{t+1} \cdot \|\nabla_{\vec{x}} f(\vec{x}_t)\| \leqslant \Delta \end{aligned}$$

First, let's ignore the side constraint and consider the unconstrained version of this optimization problem. Then, we obtain the gradient:

$$\frac{\partial}{\partial \alpha_{t+1}} \tilde{f}^2_{\vec{x}_t}\left(\vec{x}_t - \alpha_{t+1} \nabla_{\vec{x}} f(\vec{x}_t)\right) = -\nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t) + \alpha_{t+1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot \nabla_{\vec{x}} f(\vec{x}_t)$$

By setting the gradient to zero we obtain:

$$-\nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t) + \alpha_{t+1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot \nabla_{\vec{x}} f(\vec{x}_t) = \vec{0}$$

$$\alpha_{t+1} = \frac{\nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t)}{\nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot \nabla_{\vec{x}} f(\vec{x}_t)}$$

To verify that this is an optimum, consider the second derivative:

$$\frac{\partial^2}{\partial^2 \alpha_{t+1}} \tilde{f}^2_{\vec{x}_t}\left(\vec{x}_t - \alpha_{t+1} \nabla_{\vec{x}} f(\vec{x}_t)\right) = \nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot \nabla_{\vec{x}} f(\vec{x}_t)$$

Note that, if this term is negative, we can arbitrarily decrease the second-order Taylor approximation by increasing $\alpha_{t+1}$. Our only limit is our side constraint, for which we obtain:

$$\alpha_{t+1} \cdot \|\nabla_{\vec{x}} f(\vec{x}_t)\| \leqslant \Delta \quad \Longleftrightarrow \quad \alpha_{t+1} \leqslant \frac{\Delta}{\|\nabla_{\vec{x}} f(\vec{x}_t)\|}$$

Therefore, we obtain as the overall Cauchy point:

$$\alpha_{t+1} = \begin{cases} \frac{\Delta}{\|\nabla_{\vec{x}} f(\vec{x}_t)\|} & \text{if } \nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot \nabla_{\vec{x}} f(\vec{x}_t) < 0 \\ \min\{\frac{\Delta}{\|\nabla_{\vec{x}} f(\vec{x}_t)\|}, \frac{\nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t)}{\nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot \nabla_{\vec{x}} f(\vec{x}_t)}\} & \text{otherwise} \end{cases}$$

As a final alternative, one can combine both options, using the negative gradient as direction vector close to $\vec{x}_t$ where the first-order Taylor approximation is still good and interpolating to the Newton step for larger step size $\alpha_{t+1}$. This is called the *dogleg* approach. In particular, let

$$\vec{p}^1_{t+1} := -\frac{\nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla_{\vec{x}} f(\vec{x}_t)}{\nabla_{\vec{x}} f(\vec{x}_t)^T \cdot \nabla^2_{\vec{x}} f(\vec{x}_t) \cdot \nabla_{\vec{x}} f(\vec{x}_t)} \cdot \nabla_{\vec{x}} f(\vec{x}_t) \qquad \text{and}$$

$$\vec{p}^2_{t+1} := -\left(\nabla^2_{\vec{x}} f(\vec{x}_t)\right)^{-1} \cdot \nabla_{\vec{x}} f(\vec{x}_t)$$

Then, we define the dogleg vector $\vec{p}_{t+1}$ as follows.

$$\vec{p}_{t+1} = \begin{cases} \alpha_{t+1} \cdot \vec{p}^1_{t+1} & \text{if } \alpha_{t+1} \leqslant \min\{1, \frac{\Delta}{\|\vec{p}^1_{t+1}\|}\} \\ \vec{p}^1_{t+1} + (\alpha_{t+1} - 1) \cdot (\vec{p}^2_{t+1} - \vec{p}^1_{t+1}) & \text{if } 1 < \alpha_{t+1} \leqslant \min\{2, \frac{\Delta}{\|\vec{p}^2_{t+1}\|}\} \end{cases} \tag{2.10}$$

The final question regarding the trust region method is how to set the radius of the trust region, $\Delta$. The brilliant idea of the trust region approach is to choose $\Delta$ depending on how well our Taylor approximation works. If our approximation is bad, we shrink $\Delta$ and thus ensure that our approximation will work better in the next step. If our approximation is good, we increase $\Delta$ and thus can make larger jumps.

We evaluate the quality of our approximation by considering the quotient of the actual improvement in objective function value and the expected improvement according to the Taylor approximation.

$$q_{t+1} := \frac{f(\vec{x}_t) - f(\vec{x}_{t+1})}{\tilde{f}^2_{\vec{x}_t}(\vec{x}_t) - \tilde{f}^2_{\vec{x}_t}(\vec{x}_{t+1})} \tag{2.11}$$

If $q_t$ is negative, our objective function value got worse and we should reject our step and shrink $\Delta$. If $q_t$ is small (i.e. $q_t < \frac{1}{4}$), then our approximation is bad and we should shrink $\Delta$. If $q_t$ is large (i.e. $q_t > \frac{3}{4}$), we can grow more confident and increase $\Delta$. Otherwise, we leave $\Delta$ unchanged.

The overall trust region algorithm is shown in Algorithm 9.

**Algorithm 9** A generic trust-region method for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, a starting value $\vec{x}_0 \in \mathbb{R}^K$, and a gradient threshold $\epsilon > 0$.

**function** TRUST REGION(objective function $f : \mathbb{R}^K \to \mathbb{R}$, starting value $\vec{x}_0 \in \mathbb{R}^K$, threshold $\epsilon > 0$)
  $t \leftarrow 0$.
  $\Delta \leftarrow 1$.
  **while** $\|\nabla_{\vec{x}} f(\vec{x}_t)\| > \epsilon$ **do**
    Select change vector $\vec{p}_{t+1}$ via some method (e.g. Equation 2.10).
    $\vec{x}_{t+1} \leftarrow \vec{x}_t + \vec{p}_{t+1}$.
    $q_{t+1} \leftarrow$ Equation 2.11.
    **if** $q_{t+1} < 0$ **then**
      $\vec{x}_{t+1} \leftarrow \vec{x}_t$.
      $\Delta \leftarrow \|\vec{p}_{t+1}\|/4$.
    **else if** $q_{t+1} < \frac{1}{4}$ **then**
      $\Delta \leftarrow \|\vec{p}_{t+1}\|/4$.
    **else if** $q_{t+1} > \frac{3}{4}$ and $\|\vec{p}_{t+1}\| = \Delta$ **then**
      $\Delta \leftarrow \Delta \cdot 2$.
    **end if**
    $t \leftarrow t + 1$.
  **end while**
  **return** $\vec{x}_t$.
**end function**

### 2.2.2 *Constrained Optimization*

Until now we have focused on methods for unconstrained optimization. However, what can we do in case of constrained problems? We consider three approaches here, namely the barrier, the penalty, and the projection approach, all of which use an unconstrained optimizer implicitly but tweak it in order not to violate constraints.

#### The Log-Barrier Method

We consider problems which are inequality constrained, i.e. problems of the following form.

$$\begin{aligned} \min_{\vec{x} \in \mathbb{R}^K} \quad & f(\vec{x}) && (2.12) \\ \text{s.t.} \quad & g_i(\vec{x}) \geqslant 0 && \forall i \in \{1, \ldots, m\} \end{aligned}$$

The log-barrier form of such a problem is the unconstrained optimization problem.

$$\min_{\vec{x} \in \mathbb{R}^K} \quad f(\vec{x}) - \mu \cdot \sum_{i=1}^{m} \log \left[ g_i(\vec{x}) \right] \tag{2.13}$$

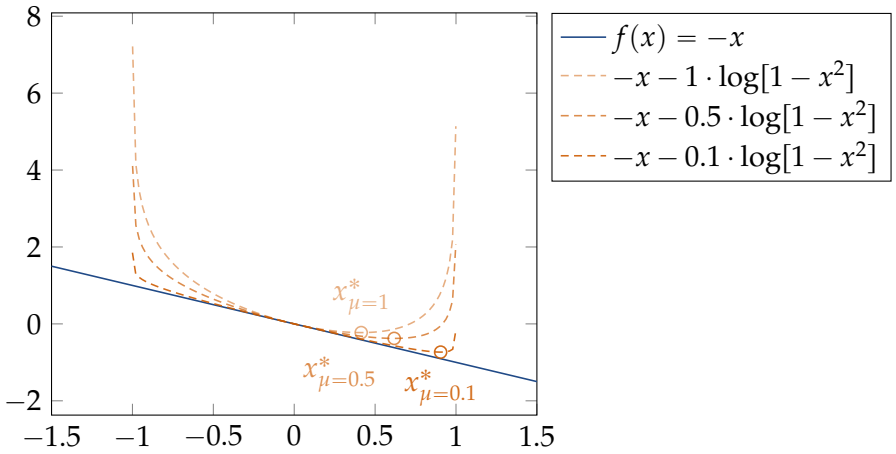



*Figure 2.4:* An illustration of the log-barrier method for Example 73. The blue, solid line shows the original objective function $f(x) = -x$, and the orange, dashed lines show the log-barrier objective function $f(x) - \mu \log[1 - x^2]$ for multiple values of $\mu$. The respective local minima are highlighted via circles.

where $\mu > 0$ is a hyper-parameter that pushes us away from the boundaries of the feasible region.

**Example 73** (Log-Barrier)**.** Consider the example problem

$$\begin{aligned} \min_{x \in \mathbb{R}} \quad & -x \\ \text{s.t.} \quad & 1 - x^2 \geqslant 0. \end{aligned}$$

The according log-barrier form 2.13 is given as follows.

$$\min_{x \in \mathbb{R}} \quad -x - \mu \cdot \log[1 - x^2]$$

Figure 2.4 shows a visualization of the original objective function and the log-barrier objective function for multiple values of $\mu$.

The idea of the log-barrier method is that any violation of the inequality constraints yields an objective function value of infinity, such that any unconstrained optimizer will be forced to stay in the feasible set. Another key advantage of the log-barrier method is that it maintains convexity, i.e. if the original problem is convex, then its log-barrier version is as well.

This does not answer a key question, though: Are the local minima of the log-barrier problem 2.13 actually the same local minima as in the primal problem 2.12? If the local minima we are far away from the boundaries of the feasible set, this is obviously the case, because then the logarithmic barrier terms have a negligible influence on the objective function. The interesting case occurs when an unconstrained local minimum is *not* the same as a constrained local minimum, i.e. if the local minimum is exactly on the boundary of the feasible set.

In more detail, assume that for our local minimum $\vec{x}^*$, the $i$th inequality constraint is active, i.e. $g_i(\vec{x}^*) = 0$. In that case, the corresponding log-barrier term $-\log[g_i(\vec{x}^*)]$ would be infinite, such that $\vec{x}^*$ is definitely *not* a local minimum of the log-barrier version of the problem. However, depending on the value of $\mu$, we will find a local minimum $\vec{x}^*_\mu$

which is *close* to $\vec{x}^*$, where the distance between $\vec{x}^*_\mu$ and $\vec{x}^*$ is smaller if $\mu$ is smaller (also refer to Figure 2.4 for an example). If we choose a very small $\mu$, then our local minima of problem 2.13 are almost the same as the local minima of problem 2.12, with an error that is bunded by $\mu \cdot m$ (Boyd and Vandenberghe 2004). For example, in Figure 2.4, our log-barrier local minimum gets closer to the true local minimum at $x^* = 1$ if we decrease $\mu$.

This analysis would tell us that we should, ideally, set $\mu$ to a very small value and then perform unconstrained optimization. Unfortunately, though, for smaller $\mu$ problem 2.13 also gets numerically much harder to solve because the gradient of the objective function close to the boundary rises very sharply. Therefore, we actually start with a reasonably large value of $\mu$ (e.g. $\mu = 1$) and then decrase $\mu$ in multiple iterations until our found local minimum of 2.12 is very close to a local minimum of 2.12. The path of local minima $\vec{x}^*_\mu$ for various values of $\mu$ is also called the *central path* (also refer to Figure 2.4 for a visualization). The final log-barrier algorithm is shown in Algorithm 10.

**Algorithm 10** The log-barrier method for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, inequality constraint functions $g_i : \mathbb{R}^K \to \mathbb{R}$, a starting value $\vec{x}_0 \in \mathbb{R}^K$, and a number of iterations $T \in \mathbb{N}$.

**function** LOG-BARRIER(objective function $f : \mathbb{R}^K \to \mathbb{R}$, inequality constraints $g_1, \dots, g_m : \mathbb{R}^K \to \mathbb{R}$, starting value $\vec{x}_0 \in \mathbb{R}^K$, iteration number $T \in \mathbb{N}$)
  $\mu \leftarrow 1$.
  **for** $t \in \{1, \dots, T\}$ **do**
    $\vec{x}_t \leftarrow \operatorname{argmin}_{\vec{x}} f(\vec{x}) - \mu \cdot \sum_{i=1}^m \log\left[g_i(\vec{x})\right]$ ▷ Unconstrained Optimization
    $\mu \leftarrow \mu/10$.
  **end for**
  **return** $\vec{x}_T$.
**end function**

### Penalty Method

Similar to the log-barrier method, the penalty method transforms a primal problem of the form 2.12 into an unconstrained problem of the following form:

$$\min_{\vec{x} \in \mathbb{R}^K} \quad f(\vec{x}) + \mu \cdot \sum_{i=1}^m \phi\left[g_i(\vec{x})\right], \tag{2.14}$$

with the key difference that $\phi : \mathbb{R} \to \mathbb{R}$ is now *not* the log function but instead any function which is 0 for positive inputs and strictly monotonously increasing with negative inputs, i.e.: $c > 0 \Rightarrow \phi(c) = 0$ and $c < c' < 0 \Rightarrow \phi(c) > \phi(c') > 0$. Examples of such functions $\phi$ are $\phi(c) = \max\{0, -c\}$ or $\phi(c) = \max\{0, -c\}^2$.

**Example 74** (Penalty Method)**.** Consider the example problem

$$\begin{aligned} \min_{x \in \mathbb{R}} \quad & -x \\ \text{s.t.} \quad & 1 - x^2 \geqslant 0 \end{aligned}$$

The according penalty form 2.14 with quadratic penalty is given as follows.

$$\min_{x \in \mathbb{R}} \quad -x + \mu \cdot \max\{0, x^2 - 1\}^2$$

Figure 2.5 shows a visualization of the original objective function and the penalty objective function for multiple values of $\mu$.

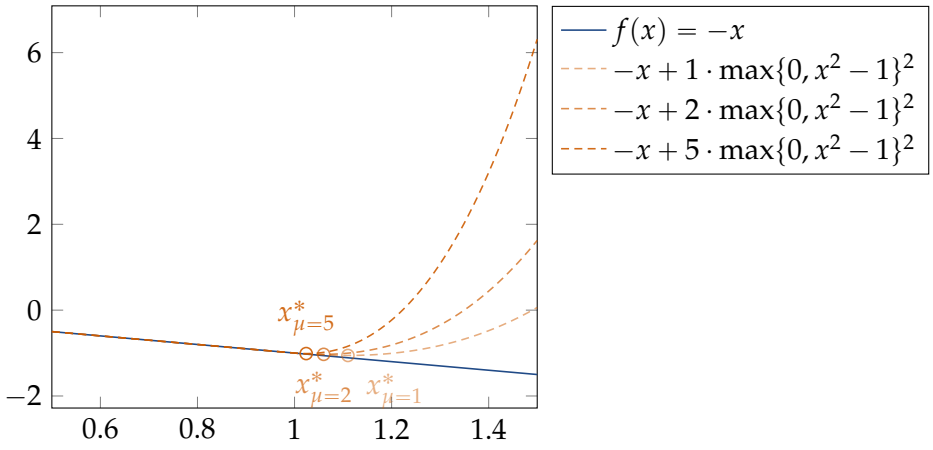



*Figure 2.5:* An illustration of the penalty method for Example 74. The blue, solid line shows the original objective function $f(x) = -x$, and the orange, dashed lines show the penalty objective function $f(x) + \mu \cdot \max\{0, 1 - x^2\}^2$ for multiple values of $\mu$. The respective local minima are highlighted via circles.

Note that the penalty method approximates the actual optimal solution similarly to the log-barrier method, but "from the other side", i.e. the solution given by the penalty method will typically be infeasible, but close to feasible. The higher the value of $\mu$, the closer we get to the feasible set, until we are approximately on the boundary. Accordingly, we obtain an algorithm 11 quite similar to the log-barrier algorithm, but increasing $\mu$ instead of decreasing it.

**Algorithm 11** The penalty method for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, inequality constraint functions $g_i : \mathbb{R}^K \to \mathbb{R}$, a starting value $\vec{x}_0 \in \mathbb{R}^K$, a penalty function $\phi : \mathbb{R} \to \mathbb{R}$, such as $\phi(c) = \max\{0, -c\}^2$, and a number of iterations $T \in \mathbb{N}$.

**function** PENALTY(objective function $f : \mathbb{R}^K \to \mathbb{R}$, inequality constraints $g_1, \ldots, g_m : \mathbb{R}^K \to \mathbb{R}$, starting value $\vec{x}_0 \in \mathbb{R}^K$, penalty function $\phi : \mathbb{R} \to \mathbb{R}$, iteration number $T \in \mathbb{N}$)
  $\mu \leftarrow 1$.
  **for** $t \in \{1, \ldots, T\}$ **do**
    $\vec{x}_t \leftarrow \operatorname{argmin}_{\vec{x}} f(\vec{x}) + \mu \cdot \sum_{i=1}^{m} \phi\big[g_i(\vec{x})\big]$ ▷ Unconstrained Optimization
    $\mu \leftarrow \mu \cdot 10$.
  **end for**
  **return** $\vec{x}_T$.
**end function**

The key advantage of the penalty method is that it yields non-infinite values across the entire domain, which makes it possible to start the optimization at any point. Accordingly, we can also use the penalty method to first find a point in the feasible set by ignoring the original objective function in our optimization entirely, and then using the log-barrier method to find the actual optimum.

### Projection Methods

Both the log-barrier method and the penalty method can handle inequality constraints, but fail to treat equality constraints appropriately because we always deviate slightly

from the boundary of the feasible set, while we require exactness on the boundary for equality constraints. Fortunately, equality constraints can be addressed in other ways. In more detail, let us consider problems of the following form:

$$\begin{aligned} \min_{\vec{x}\in\mathbb{R}^K} \quad & f(\vec{x}) \\ \text{s.t.} \quad & h_j(\vec{x}) = 0 \qquad\qquad \forall j \in \{1,\dots,n\} \end{aligned}$$

A first approach is to use the equality constraints $h_j(\vec{x})$ to solve for some of the variables $x_k$, such that we can re-write our problem in a form without equality constraints and with less variables, as we did in Example 68. This is the most elegant variant because we reduce the complexity of the problem and get rid of all equality constraints.

However, if our equality constraints can not be solved analytically for fewer variables or if the solutions are numerically unstable, we need to employ different means, such as projection methods. A projection method performs step-wise unconstrained optimization via one of our numeric methods above, but in each step ensures that the equality constraints hold again. We introduce two variants of this approach here.

First, we consider the reset projection method, where we perform a single step of an unconstrained numeric optimizer on $f(\vec{x})$ and then reset the current point $\vec{x}_t$ such that all equality constraints hold again. This is illustrated in Algorithm 12.

**Algorithm 12** The reset projection method for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, equality constraint functions $h_j : \mathbb{R}^K \to \mathbb{R}$, a starting value $\vec{x}_0 \in \mathbb{R}^K$, and some numeric optimizer $\rho : \mathbb{R}^K \to \mathbb{R}^K$, which returns a new point for each current point.

**function** RESET-PROJECTION(objective function $f : \mathbb{R}^K \to \mathbb{R}$, equality constraints $h_1, \dots, g_n : \mathbb{R}^K \to \mathbb{R}$, starting value $\vec{x}_0 \in \mathbb{R}^K$, optimizer $\rho : \mathbb{R}^K \to \mathbb{R}^K$)
  $t \leftarrow 0$.
  Reset $\vec{x}_t$ such that $h_1(\vec{x}_t) = \dots = h_n(\vec{x}_t) = 0$.
  **while** optimization criterion is not fulfilled **do**
    $\vec{x}_{t+1} \leftarrow \rho(\vec{x}_t)$. ▷ Unconstrained Optimization
    Reset $\vec{x}_{t+1}$ such that $h_1(\vec{x}_{t+1}) = \dots = h_n(\vec{x}_{t+1}) = 0$.
    $t \leftarrow t + 1$.
  **end while**
  **return** $\vec{x}_t$.
**end function**

**Example 75** (Reset projection method)**.** Consider the following problem.

$$\begin{aligned} \min_{x,y\in\mathbb{R}} \quad & -3 \cdot x - y \\ \text{s.t.} \quad & x^2 + y^2 = 1 \end{aligned}$$

If we are given any point $(x, y) \in \mathbb{R}^2$, we can ensure that our equality constraint holds by dividing both $x$ and $y$ by $\sqrt{x^2 + y^2}$.

Accordingly, if we apply Algorithm 12 with $x_0 = (1, 0)$ and $\rho(x, y) = (x, y) - 0.1 \cdot \nabla_{(x,y)} f(x, y)$, we obtain the optimization in Figure 2.6.

A second projection method is to manipulate the search direction for optimization such that it is orthogonal to the equality constraints, i.e. if we move along this search direction, the equality constraints still hold. This variant only makes sense if the feasible

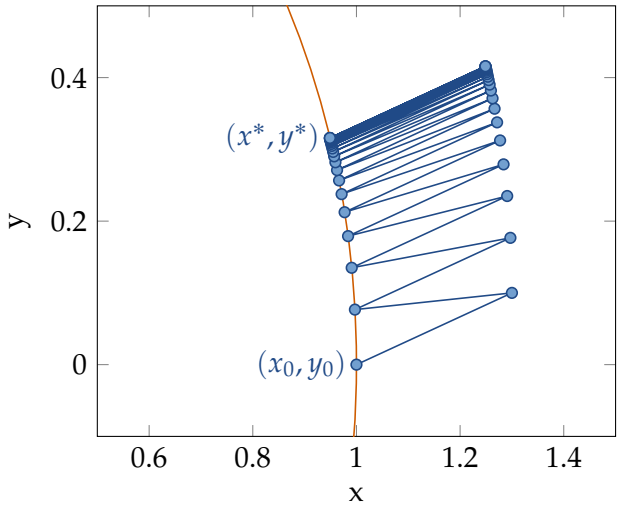



*Figure 2.6:* A visualization of reset projected gradient descent for Example 75. We start at point $(x_0, y_0) = (1, 0)$ and after each gradient step on the objective function $f(x_t, y_t) = -3 \cdot x_t - y_t$ we ensure that $x_t^2 + y_t^2 = 1$ holds by dividing both $x_t$ and $y_t$ by $\sqrt{x_t^2 + y_t^2}$. Accordingly, we observe a zig-zag line during optimization, where the point leaves the feasible set after each gradient step and re-enters the feasible set after each reset step. The feasible set is visualized as an orange circle, the points $(x_t, y_t)$ as blue circles.

set of the equality constraints is a straight line, i.e. if the equality constraints are affine. In that case, we can express our $n$ equality constraints as a matrix-vector equation $\boldsymbol{A} \cdot \vec{x} = \vec{b}$ for some $n \times K$ matrix $\boldsymbol{A}$ and some vector $\vec{b} \in \mathbb{R}^n$.

Now, assume that $\boldsymbol{A} \cdot \vec{x}_t = \vec{b}$ holds for our current point $\vec{x}_t$ and that our next point $\vec{x}_{t+1}$ is given as $\vec{x}_{t+1} = \vec{x}_t + \alpha_{t+1} \cdot \vec{p}_{t+1}$ for some search direction $\vec{p}$ and some step width $\alpha_{t+1}$ as given by a numeric optimizer.

Then, our equality constraints hold for $\vec{x}_{t+1}$ if and only if:

$$\boldsymbol{A} \cdot \vec{x}_{t+1} = \vec{b} \quad \iff \quad \boldsymbol{A} \cdot \vec{x}_t + \alpha_{t+1} \cdot \boldsymbol{A} \cdot \vec{p} = \vec{b} \quad \iff \quad \boldsymbol{A} \cdot \vec{p} = \vec{0}$$

Our challenge is that an unconstrained numeric optimizer does not necessarily return a search direction $\vec{p}$ such that $\boldsymbol{A} \cdot \vec{p} = \vec{0}$. Therefore, we need to find a search direction that is as similar as possible to $\vec{p}$ and for which this constraint holds, i.e. we need to solve the following optimization problem.

$$\begin{aligned} \min_{\tilde{p}} \quad & \frac{1}{2} \|\vec{p} - \tilde{p}\|^2 \\ \text{s.t.} \quad & \boldsymbol{A} \cdot \tilde{p} = \vec{0} \end{aligned}$$

We can solve this problem analytically via the Lagrange dual. In particular, we obtain the Lagrangian $\mathcal{L}(\tilde{p}, \vec{\lambda}) = \frac{1}{2} \|\vec{p} - \tilde{p}\|^2 - \vec{\lambda}^T \cdot \boldsymbol{A} \cdot \tilde{p}$ and the gradient:

$$\nabla_{\tilde{p}} \mathcal{L}(\tilde{p}, \vec{\lambda}) = (\tilde{p} - \vec{p}) - \boldsymbol{A}^T \cdot \vec{\lambda}$$

Note that the Hessian is the identity matrix, which means that our problem is convex.

By setting this gradient to zero we obtain:

$$(\tilde{p} - \vec{p}) - \boldsymbol{A}^T \cdot \vec{\lambda} = 0 \quad \iff \quad \tilde{p} = \boldsymbol{A}^T \cdot \vec{\lambda} + \vec{p}$$

By plugging this result into our equality constraints we obtain:

$$\boldsymbol{A} \cdot \left(\boldsymbol{A}^T \cdot \vec{\lambda} + \vec{p}\right) = \vec{0} \quad \iff \quad \vec{\lambda} = -\left(\boldsymbol{A} \cdot \boldsymbol{A}^T\right)^{-1} \cdot \boldsymbol{A} \cdot \vec{p}$$

This yields the result:

$$\tilde{p} = \left[ -\boldsymbol{A}^T \cdot \left(\boldsymbol{A} \cdot \boldsymbol{A}^T\right)^{-1} \cdot \boldsymbol{A} + \boldsymbol{I}^K \right] \cdot \vec{p} \tag{2.15}$$

Because our problem is convex, this result is a global minimum.

Using the same reasoning, we can also find an initial point which is feasible and is closest to our intended initial point. We obtain the problem:

$$\begin{aligned} \min_{\tilde{x}_0} \quad & \frac{1}{2}\|\vec{x}_0 - \tilde{x}_0\|^2 \\ \text{s.t.} \quad & \boldsymbol{A} \cdot \tilde{x}_0 = \vec{b} \end{aligned}$$

Here, we obtain the following Lagrangian and gradient:

$$\begin{aligned} \mathcal{L}(\tilde{x}_0, \vec{\lambda}) &= \frac{1}{2}\|\vec{x}_0 - \tilde{x}_0\|^2 - \vec{\lambda}^T \cdot \left[\boldsymbol{A} \cdot \tilde{x}_0 - \vec{b}\right] \\ \nabla_{\tilde{x}_0} \mathcal{L}(\tilde{x}_0, \vec{\lambda}) &= (\tilde{x}_0 - \vec{x}_0) - \boldsymbol{A}^T \cdot \vec{\lambda} \end{aligned}$$

Note that the Hessian is the identity matrix, which means that our problem is convex.

By setting this gradient to zero we obtain:

$$(\tilde{x}_0 - \vec{x}_0) - \boldsymbol{A}^T \cdot \vec{\lambda} = 0 \quad \iff \quad \tilde{x}_0 = \boldsymbol{A}^T \cdot \vec{\lambda} + \vec{x}_0$$

By plugging this result into our equality constraints we obtain:

$$\boldsymbol{A} \cdot \left(\boldsymbol{A}^T \cdot \vec{\lambda} + \vec{x}_0\right) = \vec{b} \quad \iff \quad \vec{\lambda} = -\left(\boldsymbol{A} \cdot \boldsymbol{A}^T\right)^{-1} \cdot \left(\vec{b} - \boldsymbol{A} \cdot \vec{x}_0\right)$$

This yields the result:

$$\tilde{x}_0 = -\boldsymbol{A}^T \cdot \left(\boldsymbol{A} \cdot \boldsymbol{A}^T\right)^{-1} \cdot \left(\vec{b} - \boldsymbol{A} \cdot \vec{x}_0\right) + \vec{x}_0 \tag{2.16}$$

Because our problem is convex, this result is a global minimum.

Our two results yield the overall algorithm 13.

**Example 76** (Direction projection method)**.** Consider the following problem:

$$\begin{aligned} \min_{x,y \in \mathbb{R}} \quad & (x-2)^2 + (y-1)^2 \\ \text{s.t.} \quad & -2x + y = 1 \end{aligned}$$

We can re-write our problem in the form

$$\begin{aligned} \min_{(x,y) \in \mathbb{R}^2} \quad & \frac{1}{2}\|(x,y) - (2,1)\|^2 \\ \text{s.t.} \quad & \boldsymbol{A} \cdot (x,y)^T = \vec{b} \qquad\qquad \text{where} \\ & \boldsymbol{A} = (-2,1) \quad \vec{b} = 1 \end{aligned}$$

Now, let's apply Algorithm 13 to this problem with $(x_0, y_0) = (0,0)$, $\rho(x_t, y_t) = -\nabla_{(x_t,y_t)} f(x_t, y_t)$ and a constant step width of $\alpha_t = 0.1$.

Figure 2.7 illustrates the optimization procedure in this case.

This concludes our section on numeric methods for optimization.

**Algorithm 13** The direction projection method for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, equality constraints $\boldsymbol{A} \cdot \vec{x} = \vec{b}$, a starting value $\vec{x}_0 \in \mathbb{R}^K$, and some numeric optimizer $\rho : \mathbb{R}^K \to \mathbb{R}^K$ which returns a search direction for each current point.

**function** DIRECTION-PROJECTION(objective function $f : \mathbb{R}^K \to \mathbb{R}$, equality constraint $\boldsymbol{A} \in \mathbb{R}^{n \times K}$, $\vec{b} \in \mathbb{R}^K$, starting value $\vec{x}_0 \in \mathbb{R}^K$, optimizer $\rho : \mathbb{R}^K \to \mathbb{R}^K$)
  $\boldsymbol{\Phi} \leftarrow -\boldsymbol{A}^T \cdot \left(\boldsymbol{A} \cdot \boldsymbol{A}^T\right)^{-1}$.
  $\vec{x}_0 \leftarrow \boldsymbol{\Phi} \cdot \left(\vec{b} - \boldsymbol{A} \cdot \vec{x}_0\right) + \vec{x}_0$.
  $\boldsymbol{\Phi} \leftarrow \boldsymbol{\Phi} \cdot \boldsymbol{A} + \boldsymbol{I}^K$. ▷ Equation 2.16
  $t \leftarrow 0$.
  **while** optimization criterion is not fulfilled **do**
    $\vec{p}_{t+1} \leftarrow \rho(\vec{x}_t)$. ▷ Unconstrained Optimization
    $\vec{p}_{t+1} \leftarrow \boldsymbol{\Phi} \cdot \vec{p}_{t+1}$. ▷ Equation 2.15
    $\alpha_{t+1} \leftarrow \operatorname{argmin}_\alpha f(\vec{x}_t + \alpha \cdot \vec{p}_{t+1})$.
    $\vec{x}_{t+1} \leftarrow \vec{x}_t + \alpha_{t+1} \cdot \vec{p}_{t+1}$.
    $t \leftarrow t + 1$.
  **end while**
  **return** $\vec{x}_t$.
**end function**

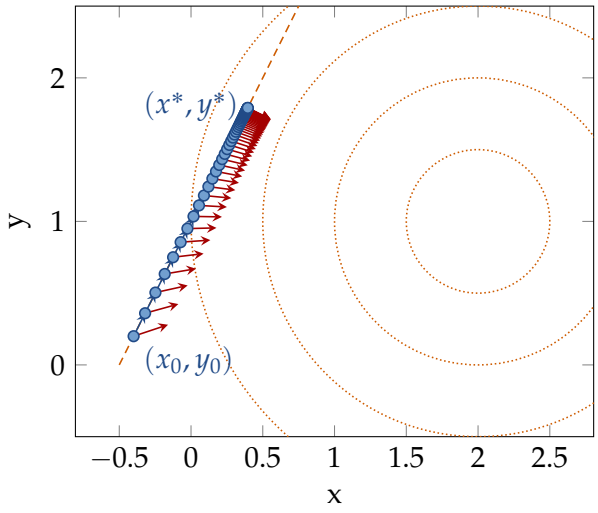



*Figure 2.7:* A visualization of direction projected gradient descent for Example 76. We start at point $(\tilde{x}_0, \tilde{y}_0) = (-0.4, 0.2)$ and before each gradient step on the objective function $f(x_t, y_t) = (x_t - 2)^2 + (y_t - 1)^2$ we ensure that the step direction maintains feasibility by applying the projection operator in Equation 2.15. Accordingly, we observe that the negative gradient directions (shown in red) are changed to directions along the feasible set (shown in blue). The contours of the objective function are shown as dotted orange lines, the feasible set is shown as dashed, orange line.

## 2.3 PROBABILISTIC OPTIMIZATION

Optimization over probabilities relies, unfortunately, on a lot of concepts from probability theory and information theory, which we can not cover in detail in these lecture notes. If you wish to learn more about this topic, a good reference is Barber (2010).

In these notes here, we will cover only two popular algorithms of probabilistic optimization, namely expectation maximization and belief propagation.

First, however, we introduce our typical objective function, namely a probability over data.

### 2.3.1 *Maximum Likelihood*

A typical problem of probabilistic optimization is to identify the probabilistic model which assigns the highest probability to a given data set. More formally, let $\mathcal{X}$ be some arbitrary set, and let $x_1, \dots, x_m \in \mathcal{X}$ be a list of samples from that set, i.e. a data set. Further, let $\mathcal{P}$ be a set of possible probability densities over $\mathcal{X}$, that is, a set of functions $p : \mathcal{X} \to \mathbb{R}$ such that $\int_{\mathcal{X}} p(x)dx = 1$. Then, we are looking for the density which assigns the highest possible probability to our data, i.e.:

$$\max_{p \in \mathcal{P}} \quad \prod_{i=1}^{m} p(x_i)$$

This is also called a *maximum likelihood* problem and we call the objective function the *likelihood*. Note that this formalization implicitly assumes that all samples are independently drawn from the same 'true' probability distribution. Otherwise, we would not be allowed to write the likelihood as a product.

How to solve such a maximum likelihood problem? In most cases, we make the problem tractable by expressing our set of probability densities $\mathcal{P}$ in a *parametric* form. In particular, we consider a family of probability densities $p_{\vec{\theta}}$, where $\vec{\theta} \in \mathbb{R}^K$ are our *parameters* and $p_{\vec{\theta}}$ is the probability density corresponding to the parameters $\vec{\theta}$. Accordingly, our maximum likelihood problem becomes:

$$\max_{\vec{\theta} \in \mathbb{R}^K} \quad \prod_{i=1}^{m} p_{\vec{\theta}}(x_i) \tag{2.17}$$

In other words, we now have a continuous optimization problem over parameters instead of an optimization problem over some function set.

**Example 77** (Exponential Distribution)**.** For example, consider the *exponential distribution*, which is defined over the positive real numbers and has the density

$$p_\lambda(x) = \lambda \cdot \exp(-\lambda \cdot x) \tag{2.18}$$

for a single parameter $\lambda \in \mathbb{R}_+$.

For a data set $x_1, \dots, x_m \in \mathbb{R}_+$, we obtain the following maximum likelihood problem.

$$\begin{aligned} \max_{\lambda \in \mathbb{R}} \quad & \prod_{i=1}^{m} \lambda \cdot \exp(-\lambda \cdot x_i) \\ \text{s.t.} \quad & \lambda > 0 \end{aligned}$$

To make a maximum likelihood problem analytically and numerically easier, we oftentimes consider the *negative log likelihood* $\ell$ instead of the likelihood, i.e. we re-write our maximum likelihood problem as follows.

$$\min_{\vec{\theta}\in\mathbb{R}^K} \quad \ell(\vec{\theta}) \quad \text{where} \quad \ell(\vec{\theta}) = -\sum_{i=1}^{m} \log\left[p_{\vec{\theta}}(x_i)\right], \tag{2.19}$$

**Example 78** (Minimum Negative Log-Likelihood for the Exponential Distribution)**.** Assume the data set $x_1, \ldots, x_m \in \mathbb{R}_+$ and consider the according minimum negative log-likelihood problem 2.19 for the exponential distribution.

$$\begin{aligned} \min_{\lambda\in\mathbb{R}} \quad & \ell(\lambda) \qquad \text{where} \quad \ell(\lambda) = -\sum_{i=1}^{m} \log[\lambda] + \log[\exp(-\lambda \cdot x_i)] = -m \cdot \log[\lambda] + \lambda \cdot \sum_{i=1}^{m} x_i \\ \text{s.t.} \quad & \lambda > 0 \end{aligned}$$

The according gradient is given as follows.

$$\nabla_\lambda \ell(\lambda) = -\frac{m}{\lambda} + \sum_{i=1}^{m} x_i$$

By setting the gradient to zero, we obtain:

$$-\frac{m}{\lambda^*} + \sum_{i=1}^{m} x_i = 0 \quad \iff \quad \lambda^* = \frac{m}{\sum_{i=1}^{m} x_i}$$

In other words, $\lambda^*$ is the inverse of the data mean.

The Hessian is given as follows.

$$\nabla^2_\lambda \ell(\lambda) = \frac{m}{\lambda^2}$$

Note that this is positive for any nonzero $\lambda$. Since our domain limits $\lambda$ to strictly positive numbers, our problem is convex. Therefore, $\lambda^*$ is a global minimum of the negative log-likelihood problem.

### 2.3.2 *Maximum a posteriori*

The maximum likelihood problem has one decisive disadvantage, namely that it can lead to models which are overly specific to our example dataset and thus do not generalize to any new data. In our exponential distribution example above, it often happens that an example data set does not contain any points that are large and thus our empiric mean of the data underestimates the 'true' mean of the 'true' underlying distribution.

An alternative to maximum likelihood optimization is offered by maximum a posteriori optimization, where we inject assumptions about our parameters. In this setting, we often express our parametrized density $p_{\vec{\theta}}(x)$ as a *conditional* density $p(x|\vec{\theta})$, such that we can express our optimization problem in Bayesian terms. In particular, given a data set $\boldsymbol{X} = (x_1, \ldots, x_m)$, we wish to optimize the *posterior* $p(\vec{\theta}|\boldsymbol{X})$, which, according to Bayes' rule, can be re-written as follows.

$$p(\vec{\theta}|\boldsymbol{X}) = \frac{p(\boldsymbol{X}|\vec{\theta}) \cdot p(\vec{\theta})}{p(\boldsymbol{X})}$$

Since $p(\boldsymbol{X})$ is not influenced by our parameter choice, we obtain the following maximum a posteriori problem.

$$\max_{\vec{\theta}\in\mathbb{R}^K} \quad \Big[\prod_{i=1}^{m} p(x_i|\vec{\theta})\Big] \cdot p(\vec{\theta}) \tag{2.20}$$

In this setting, $p(\vec{\theta})$ is our so-called *prior* and expresses our initial assumptions regarding the parameters.

**Example 79** (Maximum a posteriori for the Exponential Distribution)**.** Consider again the exponential distribution example 78, but assume now a prior $p(\lambda) = \rho \cdot \exp(-\rho \cdot \lambda)$, i.e. the $\lambda$ parameter is assumed to be itself exponentially distributed with rate parameter $\rho$. Then, the negative log of our maximum a posteriori problem is given as follows.

$$\min_{\lambda\in\mathbb{R}} \quad \ell(\lambda) \qquad \text{where} \quad \ell(\lambda) = -\log\Big[\prod_{i=1}^{m} p(x_i|\lambda)\cdot p(\lambda)\Big] = -m\cdot\log[\lambda] + \lambda\cdot\sum_{i=1}^{m} x_i + \rho\cdot\lambda$$
$$\text{s.t.} \quad \lambda > 0$$

The according gradient is given as follows.

$$\nabla_\lambda \ell(\lambda) = -\frac{m}{\lambda} + \sum_{i=1}^{m} x_i + \rho$$

By setting the gradient to zero, we obtain:

$$-\frac{m}{\lambda^*} + \sum_{i=1}^{m} x_i + 1 = 0 \quad \Longleftrightarrow \quad \lambda^* = \frac{m}{\rho + \sum_{i=1}^{m} x_i}$$

In other words, $\lambda^*$ is almost the inverse of the data mean, but is numerically more stable due to $\rho$. More precisely, $\rho > 0$ ensures that our parameter $\lambda^*$ will never degenerate to infinity. The Hessian is the same as before, such that $\lambda^*$ is indeed a global minimum of our problem.

Note that there are also more advanced Bayesian optimization schemes, which we omit here.

### 2.3.3 *Expectation Maximization*

Until now we have only considered cases where a probabilistic optimization problem can be solved analytically. What do we do if that is not the case? Either, we apply a general numeric optimizer as before, or we exploit the special structure of probabilistic optimization problems to be faster. One such fast algorithm is *Expectation Maximization (EM)* (Dempster, Laird, and Rubin 1977; Barber 2010).

EM can be applied whenever we consider a *mixture* of distributions. In particular, we assume the case where we can re-write our density $p(x)$ as follows.

$$p_{\vec{\theta}}(x) = \sum_{k=1}^{K} p_{\vec{\theta}_k}(x|k)\cdot p_{\vec{\theta}_k}(k)$$

where $p(x|k)$ is called the density for the $k$th mixture component and $p(k)$ is called the prior for the $k$th mixture component.

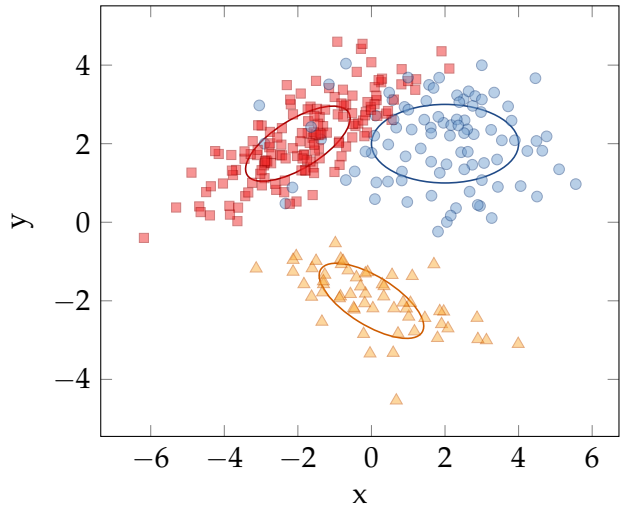



*Figure 2.8:* An illustration of a Gaussian Mixture Model (GMM) with $K = 3$ components, with the priors $\pi_1 = 0.3$, $\pi_2 = 0.2$, and $\pi_3 = 0.5$, with the means $\vec{\mu}_1 = (2,2)^T$, $\vec{\mu}_2 = (0,-2)^T$, and $\vec{\mu}_3 = (-2,2)^T$, and different covariance matrices $\boldsymbol{\Sigma}_k$. The image shows 300 samples from the probability density 2.21 and the ellipsoids indicate the area enclosed in one standard deviation of each Gaussian component.

**Example 80** (Gaussian Mixture Model (GMM))**.** Let $\mathcal{X} = \mathbb{R}^n$ be the $n$-dimensional real number space and let $K \in \mathbb{N}$. Then, the $K$-component, $n$-dimensional GMM is defined as follows.

$$p(\vec{x}) = \sum_{k=1}^{K} p(\vec{x}|k) \cdot p(k) \qquad \text{where} \qquad (2.21)$$

$$p(\vec{x}|k) = \frac{1}{\sqrt{\det(2\pi \cdot \Sigma_k)}} \cdot \exp\Big( -\frac{1}{2}(\vec{x} - \vec{\mu}_k)^T \cdot \boldsymbol{\Sigma}_k^{-1} \cdot (\vec{x} - \vec{\mu}_k)\Big),$$

$$p(k) = \pi_k,$$

where $\pi_k \in [0,1]$, $\vec{\mu}_k \in \mathbb{R}^n$, and $\boldsymbol{\Sigma}_k \in \mathbb{R}^{n \times n}$ for all $k \in \{1, \ldots, K\}$ are the parameters of the model, with the side constraints that $\sum_{k=1}^{K} \pi_k = 1$ and that $\boldsymbol{\Sigma}_k$ is positive definite.

We call $\pi_k$ the prior for the $k$th component, $\vec{\mu}_k$ the mean for the $k$th component, and $\boldsymbol{\Sigma}_k$ the covariance matrix for the $k$th component. Refer to Figure 2.8 for a two-dimensional example of a GMM.

The advantage of mixture models compared to atomic models is that they can approximate much more complicated distributions. For example, a GMM with sufficiently many mixture components can approximate any density to an arbitrary precision (Barber 2010). Unfortunately, most mixture models can not be optimized analytically. To demonstrate this fact, consider the minimum negative log-likelihood problem for a data set $x_1, \ldots, x_m$.

$$\min_{\vec{\theta}_1, \ldots, \vec{\theta}_K \in \mathbb{R}^L} \ell(\vec{\theta}_1, \ldots, \vec{\theta}_K) \quad \text{where} \quad \ell(\vec{\theta}_1, \ldots, \vec{\theta}_K) = -\sum_{i=1}^{m} \log\Big[ \sum_{k=1}^{K} p_{\vec{\theta}_k}(x_i|k) \cdot p_{\vec{\theta}_k}(k) \Big] \qquad (2.22)$$

Optimizing the log of a sum is generally not possible analytically. However, there is a neat trick that we can still apply to address problem 2.22. In particular, we will attempt

to find an upper bound for our negative log likelihood $\ell$ in Equation 2.22 and minimize the upper bound instead of $\ell$ itself. To find this upper bound, though, we first need to re-arrange our objective function quite a bit.

In a first step, we observe that our negative log likelihood $\ell$ can be re-written as follows, using the standard rules of probability theory:

$$\begin{aligned}\ell(\vec{\theta}_1,\ldots,\vec{\theta}_K) &= -\sum_{i=1}^{m}\log[p(x_i)] \\ &= -\sum_{i=1}^{m}\log\left[p_{\vec{\theta}_k}(x_i,k)/p_{\vec{\theta}_k}(k|x_i)\right] \qquad \forall k\in\{1,\ldots,K\} \\ &= \sum_{i=1}^{m}-\log\left[p_{\vec{\theta}_k}(x_i,k)\right]+\log\left[p_{\vec{\theta}_k}(k|x_i)\right] \qquad \forall k\in\{1,\ldots,K\}\end{aligned}$$

Because this equation holds for all $k\in\{1,\ldots,K\}$, it also holds if we take a weighted average over all $k$. In particular, we introduce a new $K\times m$ matrix of variables $\mathbf{\Gamma}$ with entries $\gamma_{k,i}$ for all $k\in\{1,\ldots,K\}$ and all $i\in\{1,\ldots,m\}$, which are required to be non-negative and sum up to one for all $i$, i.e.: $\sum_{k=1}^{K}\gamma_{k,i}=1$ for all $i$. A weighted average of the equation above for these variables yields:

$$\ell(\vec{\theta}_1,\ldots,\vec{\theta}_K)=\sum_{i=1}^{m}\sum_{k=1}^{K}\gamma_{k,i}\cdot\Big(-\log\left[p_{\vec{\theta}_k}(x_i,k)\right]+\log\left[p_{\vec{\theta}_k}(k|x_i)\right]\Big)$$

We can further re-write our equation as follows:

$$\begin{aligned}\ell(\vec{\theta}_1,\ldots,\vec{\theta}_K) &= -\sum_{i=1}^{m}\sum_{k=1}^{K}\gamma_{k,i}\cdot\log\left[p_{\vec{\theta}_k}(x_i,k)\right]+\sum_{k=1}^{K}\gamma_{k,i}\log\left[p_{\vec{\theta}_k}(k|x_i)\right] \\ &= \sum_{i=1}^{m}-\sum_{k=1}^{K}\gamma_{k,i}\cdot\log\left[p_{\vec{\theta}_k}(x_i,k)\right]+\sum_{k=1}^{K}\gamma_{k,i}\cdot\Big(\log\left[p_{\vec{\theta}_k}(k|x_i)\right]-\log[\gamma_{k,i}]+\log[\gamma_{k,i}]\Big) \\ &= \sum_{i=1}^{m}-\sum_{k=1}^{K}\gamma_{k,i}\cdot\log\left[p_{\vec{\theta}_k}(x_i,k)\right]+\sum_{k=1}^{K}\gamma_{k,i}\cdot\log\left[\frac{p_{\vec{\theta}_k}(k|x_i)}{\gamma_{k,i}}\right]+\sum_{k=1}^{K}\gamma_{k,i}\cdot\log[\gamma_{k,i}]\end{aligned}$$

We can interpret this equation in terms of known quantities from information theory. In particular, the second term happens to be the Kullback-Leibler divergence between $\gamma_{k,i}$ and $p_{\vec{\theta}_k}(k|x_i)$, and the third term happens to be the entropy of $\vec{\gamma}_i$. The first term is what we call *free energy* $\mathcal{Q}$. In more formal terms, we obtain:

$$\ell(\vec{\theta}_1,\ldots,\vec{\theta}_K)=\mathcal{Q}(\vec{\theta}_1,\ldots,\vec{\theta}_K,\mathbf{\Gamma})-\sum_{i=1}^{m}\mathcal{D}_{\mathrm{KL}}\big(p_{\vec{\theta}_k}(k|x_i),\vec{\gamma}_i\big)-m\cdot\mathcal{H}(\vec{\gamma}_i)\quad\text{where}\tag{2.23}$$

$$\mathcal{Q}(\vec{\theta}_1,\ldots,\vec{\theta}_K,\mathbf{\Gamma})=-\sum_{i=1}^{m}\sum_{k=1}^{K}\gamma_{k,i}\cdot\log\left[p_{\vec{\theta}_k}(x_i,k)\right],$$

$$\mathcal{D}_{\mathrm{KL}}\big(p_{\vec{\theta}_k}(k|x_i),\vec{\gamma}_i\big)=-\sum_{k=1}^{K}\gamma_{k,i}\cdot\log\left[\frac{p_{\vec{\theta}_k}(k|x_i)}{\gamma_{k,i}}\right],\text{ and}$$

$$\mathcal{H}(\vec{\gamma}_i)=-\sum_{k=1}^{K}\gamma_{k,i}\cdot\log[\gamma_{k,i}].$$

An important fact regarding this decomposition is that both the Kullback-Leibler divergence as well as the entropy are guaranteed to be non-negative. Therefore, we obtain our upper bound as desired:

$$\ell(\vec{\theta}_1, \ldots, \vec{\theta}_K) \leqslant \mathcal{Q}(\vec{\theta}_1, \ldots, \vec{\theta}_K, \mathbf{\Gamma}) \tag{2.24}$$

From our upper bound, the general EM algorithm follows almost immediately. It consists of two steps. In the expectation step, we make our bound 2.24 as tight as possible by minimizing the Kullback-Leibler divergence for all $i$. This divergence achieves its global minimum exactly at $\gamma_{k,i} = p_{\vec{\theta}_k}(k|x_i)$. In the maximization step, we set the parameters $\vec{\theta}_1, \ldots, \vec{\theta}_K$ such as to minimize $\mathcal{Q}(\vec{\theta}_1, \ldots, \vec{\theta}_K, \mathbf{\Gamma})$ while keeping all $\gamma_{k,i}$ fixed. This yields Algorithm 14.

**Algorithm 14** EM for data $x_1, \ldots, x_m$, $K$ mixture components, initial mixture parameters $\vec{\theta}_1, \ldots, \vec{\theta}_K$, and an error threshold $\epsilon$.

**function** EM(data $x_1, \ldots, x_m$, parameters $\vec{\theta}_1, \ldots, \vec{\theta}_K \in \mathbb{R}^L$, threshold $\epsilon$)
  $\gamma_{k,i} \leftarrow \frac{1}{K} \quad \forall k \in \{1, \ldots, K\}, \forall i \in \{1, \ldots, m\}$.
  **while** $\mathcal{Q}(\vec{\theta}_1, \ldots, \vec{\theta}_K, \mathbf{\Gamma}) - \ell(\vec{\theta}_1, \ldots, \vec{\theta}_K) > \epsilon$ **do**
    $\gamma_{k,i} \leftarrow p_{\vec{\theta}_k}(k|x_i) \quad \forall k \in \{1, \ldots, K\}, \forall i \in \{1, \ldots, m\}$. ▷ Expectation
    $\vec{\theta}_1, \ldots, \vec{\theta}_K \leftarrow \operatorname{argmin}_{\vec{\theta}_1, \ldots, \vec{\theta}_K} \mathcal{Q}(\vec{\theta}_1, \ldots, \vec{\theta}_K, \mathbf{\Gamma})$. ▷ Maximization
  **end while**
  **return** $\vec{\theta}_1, \ldots, \vec{\theta}_K$.
**end function**

Note that this algorithm works for *any* kind of mixture model. It is particularly popular for GMMs.

**Example 81** (EM for GMMs)**.** To apply EM to GMMs, we need two things. For the expectation step, we need a closed-form expression for the conditional probability $p(k|x_i)$. For the maximization step, we need a solution of the optimization problem

$$\min_{(\pi_1, \vec{\mu}_1, \mathbf{\Sigma}_1), \ldots, (\pi_K, \vec{\mu}_K, \mathbf{\Sigma}_K)} \quad \mathcal{Q}\big((\pi_1, \vec{\mu}_1, \mathbf{\Sigma}_1), \ldots, (\pi_K, \vec{\mu}_K, \mathbf{\Sigma}_K), \mathbf{\Gamma}\big).$$

Let's start with the expectation step. Using Bayes' rule and the rule of total probability, we can re-write our desired probability as follows:

$$p(k|\vec{x}_i) = \frac{p(\vec{x}_i|k) \cdot p(k)}{p(\vec{x}_i)} = \frac{p(\vec{x}_i|k) \cdot p(k)}{\sum_{l=1}^{K} p(\vec{x}_i, l)} = \frac{p(\vec{x}_i|k) \cdot p(k)}{\sum_{l=1}^{K} p(\vec{x}_i|l) \cdot p(l)} \tag{2.25}$$

Therefore, we can compute our expectation step using the Equation above and the definition of a GMM in Equation 2.21.

Regarding the expectation step, we obtain the following result for our free energy by plugging in Equation 2.21.

$$\begin{aligned}\mathcal{Q}\big((\pi_1, \vec{\mu}_1, \mathbf{\Sigma}_1), \ldots, (\pi_K, \vec{\mu}_K, \mathbf{\Sigma}_K), \mathbf{\Gamma}\big) &= -\sum_{i=1}^{m} \sum_{k=1}^{K} \gamma_{k,i} \cdot \log\big[p(\vec{x}_i, k)\big] \\ &= -\sum_{i=1}^{m} \sum_{k=1}^{K} \gamma_{k,i} \cdot \Big(\log\big[p(\vec{x}_i|k)\big] + \log\big[p(k)\big]\Big)\end{aligned}$$

$$= \sum_{i=1}^{m} \sum_{k=1}^{K} \gamma_{k,i} \cdot \Big( \frac{n}{2} \log[2\pi] - \frac{1}{2} \log\big[\det(\mathbf{\Sigma}_k^{-1})\big] + \frac{1}{2} (\vec{x}_i - \vec{\mu}_k)^T \cdot \mathbf{\Sigma}_k^{-1} \cdot (\vec{x}_i - \vec{\mu}_k) - \log\big[\pi_k\big] \Big)$$

Now, let us attempt to minimize this quantity for our parameters $\pi_k$, $\vec{\mu}_k$, and $\mathbf{\Sigma}_k$. We obtain the following gradient for $\vec{\mu}_k$ using the rules in Appendix 2.5.2.

$$\nabla_{\vec{\mu}_k} \mathcal{Q}\big((\pi_1, \vec{\mu}_1, \mathbf{\Sigma}_1), \ldots, (\pi_K, \vec{\mu}_K, \mathbf{\Sigma}_K), \mathbf{\Gamma}\big) = \sum_{i=1}^{m} \gamma_{k,i} \cdot \mathbf{\Sigma}_k^{-1} \cdot (\vec{\mu}_k - \vec{x}_i)$$

By setting the gradient to zero we obtain:

$$\begin{aligned} 0 &= \sum_{i=1}^{m} \gamma_{k,i} \cdot \mathbf{\Sigma}_k^{-1} \cdot (\vec{\mu}_k - \vec{x}_i) \\ \mathbf{\Sigma}_k^{-1} \cdot \vec{\mu}_k \cdot \sum_{j=1}^{m} \gamma_{k,j} &= \mathbf{\Sigma}_k^{-1} \cdot \sum_{i=1}^{m} \gamma_{k,i} \cdot \vec{x}_i \\ \vec{\mu}_k &= \frac{\sum_{i=1}^{m} \gamma_{k,i} \cdot \vec{x}_i}{\sum_{j=1}^{m} \gamma_{k,j}} \end{aligned} \tag{2.26}$$

Note that the last step is valid because $\Sigma_k^{-1}$ is positive definite and therefore invertible.

In other words, $\vec{\mu}_k$ is the weighted average of all data points, where the weight $\gamma_{k,i} / \sum_{j=1} \gamma_{k,j}$ indicates how important data point $i$ is for the $k$th mixture component.

Now, let us inspect the Hessian:

$$\nabla^2_{\vec{\mu}_k} \mathcal{Q}\big((\pi_1, \vec{\mu}_1, \mathbf{\Sigma}_1), \ldots, (\pi_K, \vec{\mu}_K, \mathbf{\Sigma}_K), \mathbf{\Gamma}\big) = \mathbf{\Sigma}_k^{-1} \cdot \sum_{i=1}^{m} \gamma_{k,i}$$

Because $\Sigma_k^{-1}$ is positive definite and $\sum_{i=1}^{m} \gamma_{k,i}$ is a non-negative number, our problem is convex with respect to $\vec{\mu}_k$, irrespective of the other parameters. Therefore, our solution above yields a global minimum with respect to $\vec{\mu}_k$.

In a next step, let us minimize the free energy with respect to $\mathbf{\Sigma}_k^{-1}$. We obtain the following matrix gradient.

$$\nabla_{\mathbf{\Sigma}_k^{-1}} \mathcal{Q}\big((\pi_1, \vec{\mu}_1, \mathbf{\Sigma}_1), \ldots, (\pi_K, \vec{\mu}_K, \mathbf{\Sigma}_K), \mathbf{\Gamma}\big) = \sum_{i=1}^{m} \gamma_{k,i} \cdot \Big( -\frac{1}{2} \mathbf{\Sigma}_k + \frac{1}{2} (\vec{x}_i - \vec{\mu}_k) \cdot (\vec{x}_i - \vec{\mu}_k)^T \Big)$$

Setting the gradient to zero yields:

$$\begin{aligned} 0 &= \sum_{i=1}^{m} \gamma_{k,i} \cdot \Big( -\frac{1}{2} \mathbf{\Sigma}_k + \frac{1}{2} (\vec{x}_i - \vec{\mu}_k) \cdot (\vec{x}_i - \vec{\mu}_k)^T \Big) \\ \mathbf{\Sigma}_k \cdot \sum_{j=1}^{m} \gamma_{k,j} &= \sum_{i=1}^{m} \gamma_{k,i} \cdot (\vec{x}_i - \vec{\mu}_k) \cdot (\vec{x}_i - \vec{\mu}_k)^T \\ \mathbf{\Sigma}_k &= \frac{\sum_{i=1}^{m} \gamma_{k,i} \cdot (\vec{x}_i - \vec{\mu}_k) \cdot (\vec{x}_i - \vec{\mu}_k)^T}{\sum_{j=1}^{m} \gamma_{k,j}} \end{aligned} \tag{2.27}$$

In other words, $\mathbf{\Sigma}_k$ is a weighted covariance matrix of the data, where the weights are the same as above. Note that this matrix is guaranteed to be positive definite and symmetric, assuming that at least $n$ linearly independent data points are in the data set.

The Hessian with respect to a matrix is not uniquely defined in the literature. But if we consider the Hessian with respect to all entries of $\Sigma_k^{-1}$ in concatenated form, we can compute the Hessian via the following Kronecker product (Fackler 2005):

$$\nabla^2_{\mathbf{\Sigma}_k^{-1}} \mathcal{Q}\big((\pi_1, \vec{\mu}_1, \mathbf{\Sigma}_1), \ldots, (\pi_K, \vec{\mu}_K, \mathbf{\Sigma}_K), \mathbf{\Gamma}\big) = \frac{1}{2}(\mathbf{\Sigma}_k \otimes \mathbf{\Sigma}_k)$$

Without going into too much detail regarding the Kronecker product, we can state that the Kronecker product of two positive definite matrices is also positive definite, such that the energy is convex with respect to $\mathbf{\Sigma}_k^{-1}$ and our solution is a global minimum.

Finally, consider the $\pi_k$ parameters. For these, we need to take the side constraints $\sum_{k=1}^K \pi_k = 1$ and $\pi_k \geqslant 0$ into account. Note that the energy term $\mathcal{Q}$ decreases monotonously with $\pi_k$. Therefore, we can generally expect $\pi_k \geqslant 0$ to be fulfilled because unconstrained optimization would yield $\pi_k \to \infty$. This leaves only $\sum_{k=1}^K \pi_k = 1$ as an active constraint. We obtain the following Lagrange dual.

$$\max_{\lambda \in \mathbb{R}} \min_{\pi_1, \ldots, \pi_K} \mathcal{L}(\pi_1, \ldots, \pi_k, \lambda) \quad \text{where}$$

$$\mathcal{L}(\pi_1, \ldots, \pi_k, \lambda) = \mathcal{Q}\big((\pi_1, \vec{\mu}_1, \mathbf{\Sigma}_1), \ldots, (\pi_K, \vec{\mu}_K, \mathbf{\Sigma}_K), \mathbf{\Gamma}\big) - \lambda \cdot \Big(\sum_{k=1}^K \pi_k - 1\Big)$$

As gradient, we obtain:

$$\frac{\partial}{\partial \pi_k} \mathcal{L}(\pi_1, \ldots, \pi_k, \lambda) = -\sum_{i=1}^m \gamma_{k,i} \cdot \frac{1}{\pi_k} - \lambda$$

Setting the gradient to zero yields:

$$\begin{aligned} 0 &= -\sum_{i=1}^m \gamma_{k,i} \cdot \frac{1}{\pi_k} - \lambda \\ \lambda \cdot \pi_k &= -\sum_{i=1}^m \gamma_{k,i} \\ \pi_k &= -\frac{\sum_{i=1}^m \gamma_{k,i}}{\lambda} \end{aligned}$$

By plugging this result into our side constraint we obtain:

$$\begin{aligned} 1 &= \sum_{k=1}^K -\frac{\sum_{i=1}^m \gamma_{k,i}}{\lambda} \\ \lambda &= -\sum_{i=1}^m \sum_{k=1}^K \gamma_{k,i} = -m \end{aligned}$$

This yields our solution:

$$\pi_k = \frac{\sum_{i=1}^m \gamma_{k,i}}{m} \tag{2.28}$$

Finally, consider the Hessian with respect to $\pi_k$.

$$\frac{\partial^2}{\partial^2 \pi_k} \mathcal{L}(\pi_1, \ldots, \pi_k, \lambda) = \sum_{i=1}^m \gamma_{k,i} \cdot \frac{1}{\pi_k^2}$$

This is positive for any nonzero $\pi_k$. Therefore, our problem is convex and our solution is a global minimum.

The resulting algorithm is shown in Algorithm 15. Figure 2.9 shows the algorithm for an example data set.

**Algorithm 15** EM for a GMM with $K$ mixture components for data $\vec{x}_1, \dots, \vec{x}_m \in \mathbb{R}^n$, initial means $\vec{\mu}_1, \dots, \vec{\mu}_K \in \mathbb{R}^n$, initial covariance matrices $\mathbf{\Sigma}_1, \dots, \mathbf{\Sigma}_K \in \mathbb{R}^{n \times n}$, initial priors $\pi_1, \dots, \pi_K \in [0,1]$ with $\sum_{k=1}^{K} \pi_k = 1$, and an error threshold $\epsilon > 0$.

**function** GMM-EM(data $\vec{x}_1, \dots, \vec{x}_m \in \mathbb{R}^n$, means $\vec{\mu}_1, \dots, \vec{\mu}_K \in \mathbb{R}^n$, covariances $\mathbf{\Sigma}_1, \dots, \mathbf{\Sigma}_K \in \mathbb{R}^{n \times n}$, priors $\pi_1, \dots, \pi_K \in [0,1]$, threshold $\epsilon > 0$)
  $\gamma_{k,i} \leftarrow \frac{1}{K} \quad \forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, m\}$.
  **while** $\mathcal{Q}\big((\pi_1, \vec{\mu}_1, \mathbf{\Sigma}_1), \dots, (\pi_K, \vec{\mu}_K, \mathbf{\Sigma}_K), \mathbf{\Gamma}\big) - \ell\big((\pi_1, \vec{\mu}_1, \mathbf{\Sigma}_1), \dots, (\pi_K, \vec{\mu}_K, \mathbf{\Sigma}_K)\big) > \epsilon$ **do**
    $\gamma_{k,i} \leftarrow$ Equation 2.25 $\forall k \in \{1, \dots, K\}, \forall i \in \{1, \dots, m\}$. ▷ Expectation
    $\vec{\mu}_k \leftarrow$ Equation 2.26 $\forall k \in \{1, \dots, K\}$. ▷ Maximization
    $\mathbf{\Sigma}_k \leftarrow$ Equation 2.27 $\forall k \in \{1, \dots, K\}$. ▷ Maximization
    $\pi_k \leftarrow$ Equation 2.28 $\forall k \in \{1, \dots, K\}$. ▷ Maximization
  **end while**
  **return** $(\pi_1, \vec{\mu}_1, \mathbf{\Sigma}_1), \dots, (\pi_K, \vec{\mu}_K, \mathbf{\Sigma}_K)$.
**end function**

### 2.3.4 *Belief Propagation and Max-Product-Algorithm*

Belief propagation is concerned with probabilistic inference over *networks* of discrete random variables. In other words, we consider cases where we have many variables at the same time, all of which can only take one of finitely many values, and where the joint probability of all variables is fully specified. The reason why we consider those variables to be connected in networks is that fully specifying a joint distribution over many variables has exponential complexity in the number of variables. We can avoid this exponential complexity by applying reasonable independence assumptions - and those assumptions can be expressed intuitively in a graph model.

For simplicity, we will restrict ourselves here to *belief networks* (refer to Barber (2010) for more general kinds of models). A belief network is, in general, a directed, acyclic graph where each node is labelled with a variable and where an edge between variable $x$ and $y$ indicates that $y$ conditionally depends on $x$. Conversely, $y$ is conditionally independent from all variables that have no edge to $y$ if conditioned on the variables that do have an edge to $y$.

More specifically, we consider belief networks which have a *tree* shape, i.e. each node has precisely one child, except for the root, which has none.

**Example 82** (Examn Network)**.** Consider the example of a written examn in school. The grade $g$ of a student in that examn may depend on the skill $s$ the student has acquired over the semester and the effort $\epsilon$ the student invests to prepare for this specific examn. Further, the skill may depend on the quality of teaching $\tau$, the prior knowledge of the student $\pi$, and the effort of the student during the semester $e$. This model is visualized as a belief network in 2.10. According to our model, the joint probability distribution over all these variables can be factorized as follows.

$$p(g, s, \tau, \pi, e, \epsilon) = p(g|s, \epsilon) \cdot p(s|\tau, \pi, e) \cdot p(\tau) \cdot p(\pi) \cdot p(e) \cdot p(\epsilon)$$

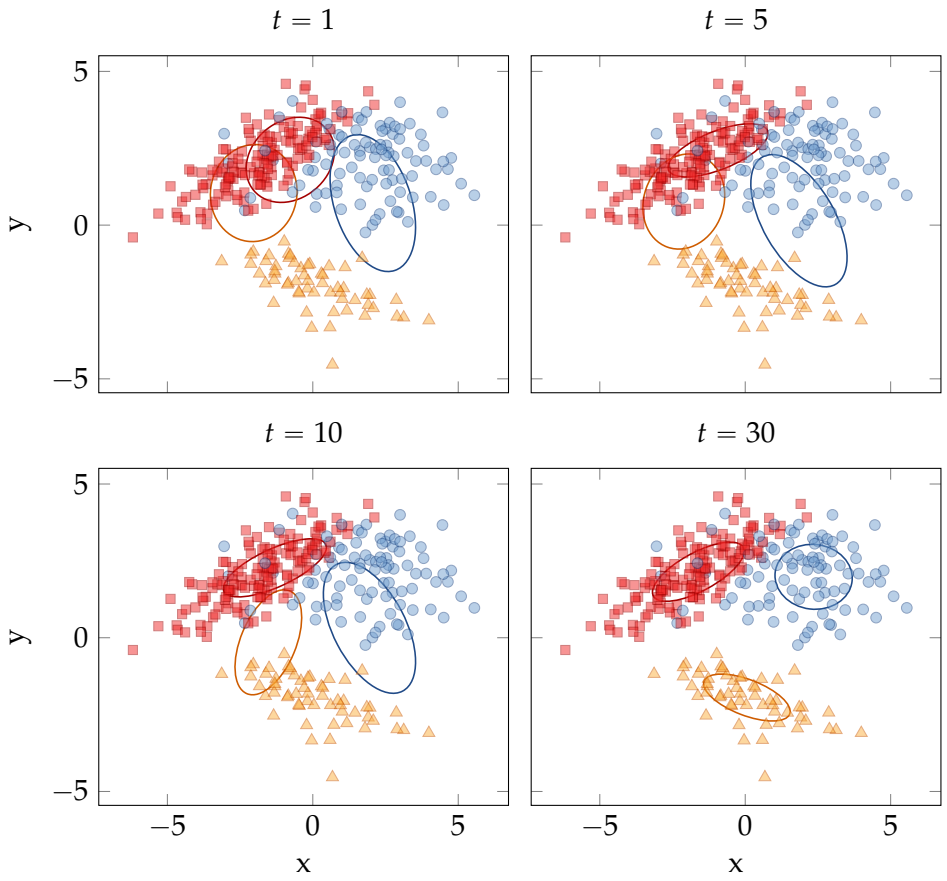



*Figure 2.9:* An illustration of the EM algorithm 15 for a GMM with $K = 3$ components on the data from Figure 2.8. Top left: After one iteration. Top right: After five iterations. Bottom left: After ten iterations. Bottom right: After thirty iterations.

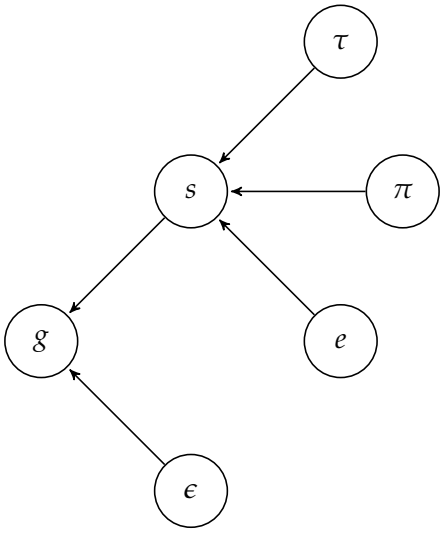



*Figure 2.10:* An example belief network with six variables $g$, $s$, $\tau$, $\pi$, $e$, and $\epsilon$, where $g$ depends on $s$ and $\epsilon$ and $s$ depends on $\tau$, $\pi$, and $e$.

Assume that all variables are in the range $\{1,2,3,4,5,6\}$, i.e. school grades. In that case, specifying the full joint probability would require $6^6 - 1 = 46655$ parameters. In our belief network, we save a lot of these parameters due to independence assumptions. In particular, the distribution $p(g|s,\epsilon)$ requires only 180 parameters, the distribution $p(s|\tau,\pi,e)$ requires 1080 parameters, and the distributions $p(\tau)$, $p(\pi)$, $p(e)$, and $p(\epsilon)$ each require 5 parameters, yielding 1280 parameters overall, a saving of over 97%.

A key optimization question over such network models is to find the most likely value for each of the variables, i.e.:

$$\max_{x_1,\ldots,x_m} \quad p(x_1,\ldots,x_m) \tag{2.29}$$

If we try to solve such a problem naively, we would have to try all possible combinations of values for all variables, which is exponential in $m$. Instead, we can exploit our factorization by means of the *max-product* algorithm. In this algorithm, we only consider all possible combinations of values for the current variable in the tree and its parents, which is typically much less than the combinations of all variables in the network.

**Example 83** (Max-Product Decomposition for the Examn network)**.** For the examn network above, we can decompose the maximization problem 2.29 as follows.

$$\begin{aligned}\max_{g,s,\tau,\pi,e,\epsilon} p(g,s,\tau,\pi,e,\epsilon) &= \max_{g,s,\tau,\pi,e,\epsilon} p(g|s,\epsilon)\cdot p(s|\tau,\pi,e)\cdot p(\tau)\cdot p(\pi)\cdot p(e)\cdot p(\epsilon)\\ &= \max_{g,s,\epsilon} p(g|s,\epsilon)\cdot p(\epsilon)\cdot \max_{s,\tau,\pi,e} p(s|\tau,\pi,e)\cdot p(\tau)\cdot p(\pi)\cdot p(e)\end{aligned}$$

The general max-product algorithm for belief trees is shown in Algorithm 16. Note that this algorithm can be extended to also return the optimal values by either storing those values during computation or by means of backtracing. Further note that we can also compute conditional maxima by restricting the domain of the variables we condition on. Note that the returned probability in these cases is incorrect, but the variable values are correct.

**Example 84** (Max-Product Algorithm for the Examn Network)**.** In our examn network, the a priori most likely values are $\tau = 2$, $\pi = 5$, $e = 3$, $s = 3$, $\epsilon = 3$, and $g = 3$.

However, if we condition on $\tau \in [4,5,6]$, $s$ and $g$ change to 4 respectively, i.e. if the teaching is worse, we also expect less skill and worse grades.

**Algorithm 16** The max-product algorithm for belief trees with variables $x_1, \ldots, x_m$, each of which has domain $\mathcal{X}(x_i)$ with finite size and only depends on its respective parents $\mathcal{P}(x_i)$. Note that this version of the algorithm does *not* yet return the optimal variable values. For that purpose, we need a backtracing scheme.

**function** MAX-PRODUCT(random variables $x_1, \ldots, x_m$, domains $\mathcal{X}(x_1), \ldots, \mathcal{X}(x_m)$, and parents $\mathcal{P}(x_1), \ldots, \mathcal{P}(x_m)$, start variable $x$)
    **if** $\mathcal{P}(x)$ is empty **then**
        **return** $\max_{v \in \mathcal{X}(x)} p(v)$.
    **end if**
    $x_1, \ldots, x_n \leftarrow \mathcal{P}(x)$.
    $p_{\max} \leftarrow 0$.
    **for** all value combinations $(v_1, \ldots, v_n) \in \mathcal{X}(x_1) \times \ldots \times \mathcal{X}(x_n)$ **do**
        **for** $i \leftarrow 1, \ldots, n$ **do**
            Restrict $\mathcal{X}(x_i) \leftarrow \{v_i\}$.
        **end for**
        $p \leftarrow 1$.
        **for** $i \leftarrow 1, \ldots, n$ **do**
            $p \leftarrow p \cdot$ MAX-PRODUCT$(x_1, \ldots, x_m, \mathcal{X}(x_1), \ldots, \mathcal{X}(x_m), \mathcal{P}(x_1), \ldots, \mathcal{P}(x_m), x_i)$
        **end for**
        $p \leftarrow \max_{v \in \mathcal{X}(x)} p(v|v_1, \ldots, v_n) \cdot p$.
        **if** $p > p_{\max}$ **then**
            $p_{\max} \leftarrow p$.
        **end if**
    **end for**
    **return** $p_{\max}$.
**end function**

## 2.4 CONVEX PROGRAMMING

In the previous section we have covered specialized algorithms if our optimization problem is probabilistic. Another special case are convex optimization problems. Recall from Section 1.3 that for convex problems we can easily find a local minimum via numeric approaches and such an local minimum is guaranteed to be a global minimum. Therefore, even our standard numeric approaches above will automatically be more efficient for convex problems. Beyond this, there are specific algorithms which are even faster for certain subclasses of convex optimization problems. Two of these interesting subclasses are *linear* and *quadratic* programs, which we cover here.

Unfortunately, the many tricks involved to make such algorithms fast are too involved to cover here. However, we will still discuss the specific form of these problems and how to transform an optimization problem in a linear or quadratic form.

### 2.4.1 *Linear Programming*

A *linear* program is an optimization problem of the following form.

$$\begin{aligned} \min_{\vec{x}\in\mathbb{R}^K} \quad & \vec{c}^T \cdot \vec{x} \\ \text{s.t.} \quad & \boldsymbol{A} \cdot \vec{x} \leqslant \vec{b} \\ & \boldsymbol{A}^{\text{eq}} \cdot \vec{x} = \vec{b}^{\text{eq}} \\ & \vec{lb} \leqslant \vec{x} \leqslant \vec{ub}, \end{aligned} \tag{2.30}$$

where $\vec{c} \in \mathbb{R}^K$, $\boldsymbol{A} \in \mathbb{R}^{m\times K}$, $\vec{b} \in \mathbb{R}^m$, $\boldsymbol{A}^{\text{eq}} \in \mathbb{R}^{n\times K}$, $\vec{b}^{\text{eq}} \in \mathbb{R}^n$, $\vec{lb} \in \mathbb{R}^K$, and $\vec{ub} \in \mathbb{R}^K$. Expressing the linear program in this form has the advantage that most libraries accept the parameters as written here as input.

**Example 85** (Nutritional Food)**.** Assume you plan your meals for the following day and want to ensure that you get all nutrients you need. Further assume that you consider $m$ different nutrients overall and that you have a cookbook with $K$ recipes that describes the amount $a_{i,j}$ of nutrient $j$ per portion of dish $i$ as well as the time $c_i$ needed to cook one portion of dish $i$. Further, assume that you need a daily dose of at least $b_j$ of nutrient $j$. Then, the problem of selecting what to cook such that you get all nutrients you need but need the least amount of cooking time is the following linear problem:

$$\begin{aligned} \min_{\vec{x}\in\mathbb{R}^K} \quad & \vec{c}^T \cdot \vec{x} \\ \text{s.t.} \quad & \boldsymbol{A} \cdot \vec{x} \leqslant \vec{b} \\ & \vec{0} \leqslant \vec{x} \leqslant \vec{\infty}, \end{aligned} \tag{2.31}$$

### 2.4.2 *Quadratic Programming*

A *quadratic* program is an optimization problem of the following form.

$$\begin{aligned} \min_{\vec{x}\in\mathbb{R}^K} \quad & \frac{1}{2}\vec{x}^T \cdot \boldsymbol{H} \cdot \vec{x}\vec{c}^T \cdot \vec{x} \\ \text{s.t.} \quad & \boldsymbol{A} \cdot \vec{x} \leqslant \vec{b} \end{aligned} \tag{2.32}$$

$$\boldsymbol{A}^{\text{eq}} \cdot \vec{x} = \vec{b}^{\text{eq}}$$
$$\vec{lb} \leqslant \vec{x} \leqslant \vec{ub},$$

where $\boldsymbol{H}$ is a positive semi-definite matrix, $\vec{c} \in \mathbb{R}^K$, $\boldsymbol{A} \in \mathbb{R}^{m \times K}$, $\vec{b} \in \mathbb{R}^m$, $\boldsymbol{A}^{\text{eq}} \in \mathbb{R}^{n \times K}$, $\vec{b}^{\text{eq}} \in \mathbb{R}^n$, $\vec{lb} \in \mathbb{R}^K$, and $\vec{ub} \in \mathbb{R}^K$. Note that $\boldsymbol{H}$ is also the Hessian of this problem, such that positive semi-definiteness is required for convexity.

**Example 86** (Gaussian $\epsilon$-likelihood problem)**.** Assume we are given a set of data points $\vec{x}_1, \ldots, \vec{x}_m \in \mathbb{R}^K$ and wish to find the Gaussian distribution with a pre-defined covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{K \times K}$, such that even the least likely data point is still as likely as possible. Accordingly, the only parameter left to find is the mean of the distribution, and we can formalize this problem as follows.

$$\max_{\vec{\mu} \in \mathbb{R}^K} \quad \min_{i \in \{1,\ldots,m\}} \quad p(\vec{x}_i | \vec{\mu}, \boldsymbol{\Sigma})$$

where $p(\vec{x}_i | \vec{\mu}, \boldsymbol{\Sigma})$ is the density function for the Gaussian with mean $\vec{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.

Using a helping variable $\epsilon$, we can re-write this max-min problem alternatively as follows.

$$\begin{aligned} \max_{\vec{\mu} \in \mathbb{R}^K, \epsilon \in \mathbb{R}} \quad & \epsilon \\ \text{s.t.} \quad & p(\vec{x}_i | \vec{\mu}, \boldsymbol{\Sigma}) \geqslant \epsilon && \forall i \in \{1, \ldots, m\} \end{aligned}$$

In other words, we try to find a mean $\vec{\mu}$, such that every data point $\vec{x}_i$ has a likelihood of at least $\epsilon$ and $\epsilon$ is maximized.

Now, recall that $p(\vec{x}_i | \vec{\mu}, \boldsymbol{\Sigma})$ has the following form.

$$p(\vec{x}_i | \vec{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{\det(2\pi \cdot \boldsymbol{\Sigma})}} \cdot \exp\Big( -\frac{1}{2} (\vec{x}_i - \vec{\mu})^T \cdot \boldsymbol{\Sigma}^{-1} \cdot (\vec{x}_i - \vec{\mu}) \Big)$$

Accordingly, we can re-write our inequality constraints as follows.

$$\begin{aligned} & p(\vec{x}_i | \vec{\mu}, \boldsymbol{\Sigma}) \geqslant \epsilon && \forall i \in \{1, \ldots, m\} \\ \iff & \log\big[ p(\vec{x}_i | \vec{\mu}, \boldsymbol{\Sigma}) \big] \geqslant \log[\epsilon] && \forall i \in \{1, \ldots, m\} \\ \iff & -\frac{1}{2} \log\big[ \det(2\pi \cdot \boldsymbol{\Sigma}) \big] - \frac{1}{2} (\vec{x}_i - \vec{\mu})^T \cdot \boldsymbol{\Sigma}^{-1} \cdot (\vec{x}_i - \vec{\mu}) \geqslant \log[\epsilon] && \forall i \in \{1, \ldots, m\} \end{aligned}$$

Now, we define a new variable $r$ as follows.

$$r^2 := -2 \log[\epsilon] - \log\big[ \det(2\pi \cdot \boldsymbol{\Sigma}) \big]$$

Accordingly, we can re-write our side constraints as follows.

$$\begin{aligned} & \frac{1}{2} r^2 - \frac{1}{2} (\vec{x}_i - \vec{\mu})^T \cdot \boldsymbol{\Sigma}^{-1} \cdot (\vec{x}_i - \vec{\mu}) \geqslant 0 && \forall i \in \{1, \ldots, m\} \\ \iff & r^2 - (\vec{x}_i - \vec{\mu})^T \cdot \boldsymbol{\Sigma}^{-1} \cdot (\vec{x}_i - \vec{\mu}) \geqslant 0 && \forall i \in \{1, \ldots, m\} \end{aligned}$$

Further note that $r^2$ strictly monotonously decreases for rising $\epsilon$ (and vice versa), such that we can re-write the entire problem as follows.

$$\min_{\vec{\mu} \in \mathbb{R}^K, r \in \mathbb{R}} \quad r^2$$

$$\text{s.t.} \quad r^2 - (\vec{x}_i - \vec{\mu})^T \cdot \mathbf{\Sigma}^{-1} \cdot (\vec{x}_i - \vec{\mu}) \geqslant 0 \qquad \forall i \in \{1, \dots, m\}$$

Note that this is not yet a quadratic problem because our side constraints are nonlinear. However, we can transform it into a quadratic problem by employing the Wolfe dual.

$$\begin{aligned} \max_{\vec{\lambda} \in \mathbb{R}^m, \vec{\mu} \in \mathbb{R}^K, r \in \mathbb{R}} \quad & r^2 - \sum_{i=1}^{m} \lambda_i \cdot \left(r^2 - (\vec{x}_i - \vec{\mu})^T \cdot \mathbf{\Sigma}^{-1} \cdot (\vec{x}_i - \vec{\mu})\right) \\ \text{s.t.} \quad & \vec{\lambda} \geqslant 0 \\ & \frac{\partial}{\partial r} r^2 = \frac{\partial}{\partial r} \sum_{i=1}^{m} \lambda_i \cdot \left(r^2 - (\vec{x}_i - \vec{\mu})^T \cdot \mathbf{\Sigma}^{-1} \cdot (\vec{x}_i - \vec{\mu})\right) \\ & \nabla_\mu r^2 = \nabla_\mu \sum_{i=1}^{m} \lambda_i \cdot \left(r^2 - (\vec{x}_i - \vec{\mu})^T \cdot \mathbf{\Sigma}^{-1} \cdot (\vec{x}_i - \vec{\mu})\right) \end{aligned}$$

The two equality constraints evaluate to the following equations.

$$2r = \sum_{i=1}^{m} \lambda_i \cdot (2r) \quad \Longleftrightarrow \quad \sum_{i=1}^{m} \lambda_i = 1$$

$$\vec{0} = \sum_{i=1}^{m} \lambda_i \cdot 2\mathbf{\Sigma}^{-1} \cdot (\vec{\mu} - \vec{x}_i) \quad \Longleftrightarrow \quad \vec{\mu} = \sum_{i=1}^{m} \lambda_i \cdot \vec{x}_i = \boldsymbol{X} \cdot \vec{\lambda}$$

where $\boldsymbol{X} = (\vec{x}_1, \dots, \vec{x}_m)$.

Accordingly, we can incorporate our side-constraints into our objective function, which yields:

$$\begin{aligned} f(\vec{\lambda}, \vec{\mu}, r^2) &= r^2 - \sum_{i=1}^{m} \lambda_i \cdot \left(r^2 - (\vec{x}_i - \boldsymbol{X} \cdot \vec{\lambda})^T \cdot \mathbf{\Sigma}^{-1} \cdot (\vec{x}_i - \boldsymbol{X} \cdot \vec{\lambda})\right) \\ &= r^2 - r^2 \cdot \underbrace{\left(\sum_{i=1}^{m} \lambda_i\right)}_{=1} + \sum_{i=1}^{m} \lambda_i \cdot \underbrace{\vec{x}_i^T \cdot \mathbf{\Sigma}^{-1} \cdot \vec{x}_i}_{-c_i :=} - 2 \cdot \underbrace{\left(\sum_{i=1}^{m} \lambda_i \cdot \vec{x}_i^T\right)}_{= \vec{\lambda}^T \cdot \boldsymbol{X}^T} \cdot \mathbf{\Sigma}^{-1} \cdot \boldsymbol{X} \cdot \vec{\lambda} \\ &\quad + \underbrace{\left(\sum_{i=1}^{m} \lambda_i\right)}_{=1} \cdot \vec{\lambda}^T \cdot \boldsymbol{X}^T \cdot \mathbf{\Sigma}^{-1} \cdot \boldsymbol{X} \cdot \vec{\lambda} \\ &= -2\vec{\lambda}^T \cdot \boldsymbol{X}^T \cdot \mathbf{\Sigma}^{-1} \cdot \boldsymbol{X} \cdot \vec{\lambda} + \vec{\lambda}^T \cdot \boldsymbol{X}^T \cdot \mathbf{\Sigma}^{-1} \cdot \boldsymbol{X} \cdot \vec{\lambda} - \vec{c}^T \cdot \vec{\lambda} \\ &= -\vec{\lambda}^T \cdot \boldsymbol{X}^T \cdot \mathbf{\Sigma}^{-1} \cdot \boldsymbol{X} \cdot \vec{\lambda} - \vec{c}^T \cdot \vec{\lambda} \end{aligned}$$

This finally leaves us with the following quadratic program.

$$\begin{aligned} \min_{\vec{\lambda} \in \mathbb{R}^m} \quad & \frac{1}{2} \cdot \vec{\lambda}^T \cdot \left(2\boldsymbol{X}^T \cdot \mathbf{\Sigma}^{-1} \cdot \boldsymbol{X}\right) \cdot \vec{\lambda} + \vec{c}^T \cdot \vec{\lambda} \\ \text{s.t.} \quad & \vec{\lambda} \geqslant 0 \end{aligned}$$

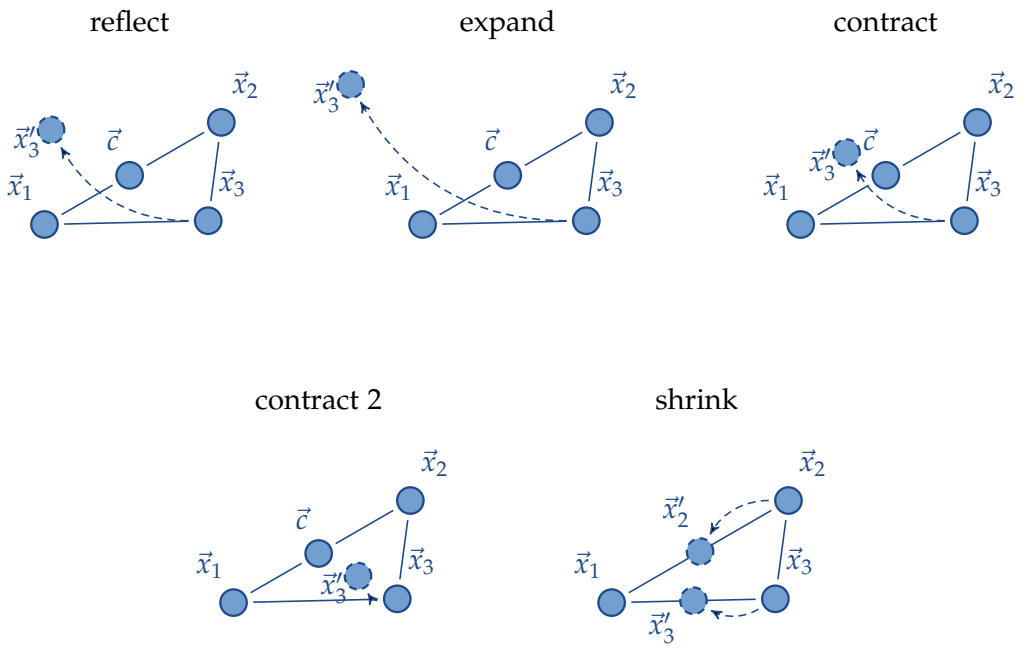



*Figure 2.11:* An illustration of the possible moves in the Nelder-Mead algorithm in two dimensions. We try the moves in the order from left to right, where the second move is tried if the first succeeds and all successive moves are tried if the one before failed. $\vec{x}_1$ marks the currently best vertex, $\vec{x}_2$ the medium one, and $\vec{x}_3$ the worst one. $\vec{c}$ is the average of $\vec{x}_1$ and $\vec{x}_2$.

## 2.5 HEURISTICS

An algorithm is called a *heuristic* if it comes with no or only weak correctness guarantees but yields a reasonably good solution reasonably quick. In optimization, heuristics are generally slower to converge and yield worse outcomes compared to the previous techniques. However, they are also more flexible and can be applied as a last resort whenever the other techniques fail, for example if no gradient information is available or in case of discrete problems.

### 2.5.1 *Gradient-free Optimization*

The first class of heuristics we cover are gradient-free optimization techniques for problems where the objective function may technically be continuous, or at least somewhat smooth, but computing the gradient is infeasible. An example is the selection of hyper-parameters for optimization problems, e.g. in machine learning. To compute the gradient with respect to hyper-parameters, we would need to compute a gradient of the entire optimization process, which is generally hard to do.

#### Downhill-Simplex / Nelder-Mead algorithm

The algorithm of Nelder and Mead (1965), also known as the downhill-simplex algorithm, optimizes a function by constructing a $K+1$ dimensional simplex (e.g. a triangle in 2D) and iteratively switches the position of the currently worst vertex until the simplex reaches a local minimum. In more detail, we try to move our worst vertex such that

it becomes better than the second-worst vertex. Our first move is to *reflect* the worst vertex at the center (see Figure 2.11, top left). If that improves our objective function even beyond the currently best vertex, we grow more confident and try to move twice as far in the same direction, i.e. we *expand* our simplex (see Figure 2.11, top center). Conversely, if our reflection does not even improve the objective function beyond the second-worst vertex, we try to move only half as far, i.e. we *contract* our simplex (see Figure 2.11, top right). If even that does not improve the objective function beyond the second-worst vertex, we try to locate our new vertex half-way between its old position and the center, i.e. an alternative contraction move (see Figure 2.11, bottom left). If that fails as well, we shrink our entire simplex toward the currently best vertex (see Figure 2.11, bottom right). The resulting algorithm is shown in Algorithm 17.

**Algorithm 17** The downhill-simplex a.k.a. Nelder-Mead algorithm for the objective function $f : \mathbb{R}^K \to \mathbb{R}$ and an error threshold $\epsilon > 0$, starting from an initial simplex $\vec{x}_1, \ldots, \vec{x}_{K+1} \in \mathbb{R}^K$.

**function** DOWNHILL-SIMPLEX(objective function $f : \mathbb{R}^K \to \mathbb{R}$, initial simplex $\vec{x}_1, \ldots, \vec{x}_{K+1} \in \mathbb{R}^K$, threshold $\epsilon > 0$.)
  Sort $\vec{x}_1, \ldots, \vec{x}_{K+1}$ ascendingly according to $f(\vec{x}_k)$.
  **while** $f(\vec{x}_{K+1}) - f(\vec{x}_1) > \epsilon$ **do**
    $\vec{c} \leftarrow \frac{1}{K} \cdot \sum_{k=1}^{K} \vec{x}_k$ ▷ Mean of $K$ best vertices
    $\vec{\delta} \leftarrow \vec{c} - \vec{x}_{K+1}$.
    **if** $f(\vec{c} + \vec{\delta}) < f(\vec{x}_K)$ **then**
      **if** $f(\vec{c} + \vec{\delta}) < f(\vec{x}_1)$ and $f(\vec{c} + 2\vec{\delta}) < f(\vec{x}_1)$ **then**
        $\vec{x}_{K+1} \leftarrow \vec{c} + 2\vec{\delta}$. ▷ Expand
      **else**
        $\vec{x}_{K+1} \leftarrow \vec{c} + \vec{\delta}$. ▷ Reflect
      **end if**
    **else if** $f(\vec{c} + \frac{1}{2}\vec{\delta}) < f(\vec{x}_K)$ **then**
      $\vec{x}_{K+1} \leftarrow \vec{c} + \frac{1}{2}\vec{\delta}$. ▷ Contract
    **else if** $f(\vec{c} - \frac{1}{2}\vec{\delta}) < f(\vec{x}_K)$ **then**
      $\vec{x}_{K+1} \leftarrow \vec{c} - \frac{1}{2}\vec{\delta}$. ▷ Contract 2
    **else**
      **for** $k \in \{2, \ldots, K+1\}$ **do**
        $\vec{x}_k \leftarrow \frac{1}{2}(\vec{x}_k + \vec{x}_1)$. ▷ Shrink
      **end for**
    **end if**
    Sort $\vec{x}_1, \ldots, \vec{x}_{K+1}$ ascendingly according to $f(\vec{x}_k)$.
  **end while**
  **return** $\vec{x}_1$.
**end function**

A key challenge in applying the downhill-simplex algorithm is initialization. If we choose the initial simplex too small, we may get stuck in a bad local optimum. If we choose it too wide, it may need a long time to converge. In general, the initial simplex must be chosen problem-dependent. If we have, for example, a good initial guess $\vec{x}_0$ and know a standard deviation $\sigma_k$ for each variable, we can sample the initial vertices of the simplex from a Gaussian with mean $\vec{x}_0$ and covariance matrix $\mathrm{diag}(\sigma_1^2, \ldots, \sigma_K^2)$.

### CMA-ES

Whenever the number of dimensions is large, the downhill-simplex method is infeasible because the number of vertices scale with the dimensionality and the number of necessary vertex movements for optimization may thus rise dramatically. An alternative to downhill-dimplex optimization is provided by *evolutionary strategies* (ES), which can be sketched as follows.

First, generate a sample of $\lambda \in \mathbb{N}$ random points from the search domain. Second, select from these the $\mu < \lambda$ points with the lowest objective function value. Third, adjust your generative process according to these selected points and continue at step one. Note that $\lambda$ and $\mu$ are hyper-parameters of this method which the user has to set beforehand.

We can distinguish two kinds of evolutionary strategies based on their selection pool in step two, namely $\lambda, \mu$-strategies if we select only from newly generated points and $\lambda + \mu$-strategies if we select from the the union of the newly generated points *and* the previous generation.

Further, we can distinguish evolutionary strategies according to the generative process for step one and how this process is adapted in step three. The probably easiest generative process is to sample our data from an isotropic Gaussian distribution with constant variance $\sigma^2 \in \mathbb{R}$ in step one and to adjust the mean of that Gaussian in step three. The resulting Algorithm is shown in Algorithm 18. Figure 2.12 (top) shows the first iterations of this algorithm on the Rosenbrock function.

**Algorithm 18** An isotropic Gaussian $\mu, \lambda$-evolutionary strategy (ES) for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, the initial mean $\vec{m}_0 \in \mathbb{R}^K$, the variance $\sigma^2 > 0$, a number of samples $\mu \in \mathbb{N}$, a number of survivors $\lambda < \mu$, and a threshold $\epsilon > 0$.

1: **function** ES(objective function $f : \mathbb{R}^K \to \mathbb{R}$, initial mean $\vec{m}_0 \in \mathbb{R}^K$, variance $\sigma^2 > 0$, sample number $\mu \in \mathbb{N}$, survivor number $\lambda < \mu$, threshold $\epsilon > 0$.)
2: $t \leftarrow 0$.
3: **while** $t < 1$ or $f(\vec{m}_t) - f(\vec{x}_{t-1}) > \epsilon$ **do**
4: Sample $\vec{x}_1, \ldots, \vec{x}_\mu$ from the Gaussian distribution with mean $\vec{m}_t$ and covariance matrix $\sigma^2 \cdot \boldsymbol{I}^K$.
5: Sort $\vec{x}_1, \ldots, \vec{x}_\mu$ ascendingly according to the objective function value $f(\vec{x}_i)$.
6: $\vec{m}_{t+1} \leftarrow \frac{1}{\lambda} \cdot \sum_{i=1}^{\lambda} \vec{x}_i$.
7: $t \leftarrow t + 1$.
8: **end while**
9: **return** $\vec{x}_1$.
10: **end function**

An issue with Algorithm 18 is that it does not adapt its step size. This way it can neither jump over long stretches with small changes in the objective function, nor can it zoom in on small changes when it is close to the optimum. To adapt the step size, we can choose to not only adapt the mean in each iteration, but also the *covariance matrix*, which then yields covariance matrix adaptation evolutionary strategy (CMA-ES). In its simplest form, CMA-ES would initialize the covariance matrix as $\sigma^2 \cdot \boldsymbol{I}^K$ as before, but adapt it in each iteration as follows.

$$\boldsymbol{\Sigma}_{t+1} \leftarrow \frac{1}{\lambda} \cdot \sum_{i=1}^{\lambda} (\vec{x}_i - \vec{m}_t) \cdot (\vec{x}_i - \vec{m}_t)^T,$$

i.e. we estimate the covariance matrix in the next step using the empirical covariance matrix of the data, as we did with the mean. Unfortunately, though, this estimate is

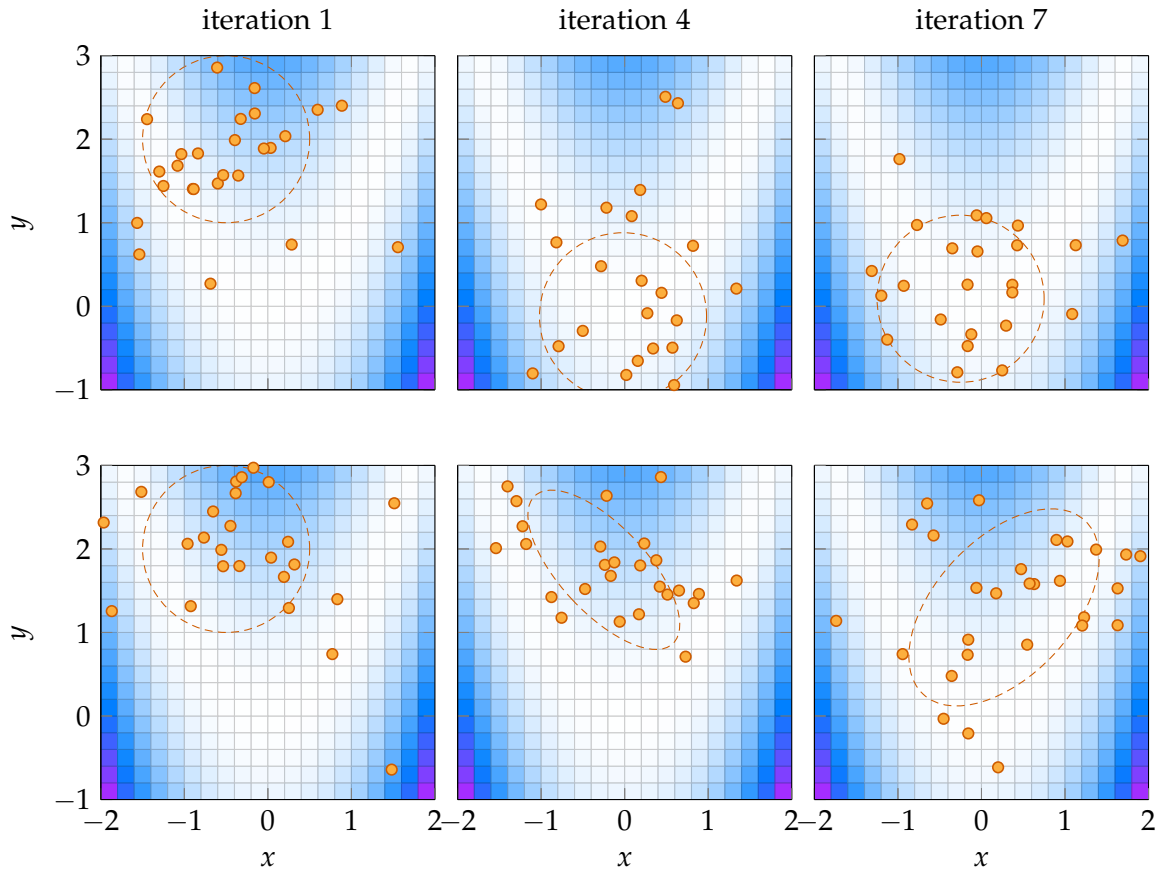



*Figure 2.12:* The first iterations of an evolutionary strategy with an isotropic Gaussian (top) and CMA-ES (bottom) with $\mu = 30$ and $\lambda = 5$. The Gaussian distribution in the respective iteration is indicated by the orange, dashed line at one standard deviation.

inaccurate in higher dimensions $K$ because the number of parameters in the covariance matrix scales quadratically with $K$ but we have only $\lambda$ points to estimate these parameters. Therefore, CMA-ES employs two tricks to estimate $\mathbf{\Sigma}$ more reliably. First, it keeps information from previous iterations and second, it adds a one-rank update which depends on the difference between the means from the current iteration and the previous iteration. In more detail, the covariance matrix representation is distributed into a scalar variance $\sigma_t^2$ and an actual matrix $\mathbf{\Sigma}$ which are updated separately.

Another innovation in CMA-ES is that not all data points are included equally in estimating mean and covariance, but rather weighted according to their objective function value. The details of the entire approach are rather involved such that we avoid a detailed discussion here. For interested readers, we point to the pseudocode in Algorithm 19. Figure 2.12 (bottom) displays some iterations of CMA-ES on the Rosenbrock function.

### Bayesian Optimization

For all optimization procedures up to this point we have assumed that we want to converge fast but that we can, essentially, perform as many evaluations of our objective function as we need. However, what if our objective function is costly to compute, e.g. because it is the solution of a subsequent optimization problem, such as a deep learning

**Algorithm 19** CMA-ES for the objective function $f : \mathbb{R}^K \to \mathbb{R}$, the initial mean $\vec{m}_0 \in \mathbb{R}^K$, the initial variance $\sigma_0^2 > 0$, a number of samples $\mu \in \mathbb{N}$, a number of survivors $\lambda < \mu$, and a threshold $\epsilon > 0$.

**function** CMA-ES(objective function $f : \mathbb{R}^K \to \mathbb{R}$, initial mean $\vec{m}_0 \in \mathbb{R}^K$, variance $\sigma_0^2 > 0$, sample number $\mu \in \mathbb{N}$, survivor number $\lambda < \mu$, threshold $\epsilon > 0$.)
  $\vec{\Sigma}_0 \leftarrow \boldsymbol{I}^K$.
  $\vec{w} \leftarrow 2(\lambda, \lambda - 1, \ldots, 1)^T / ((\lambda + 1) \cdot \lambda)$. ▷ descending weights
  $\mu_{\vec{w}} \leftarrow 1 / \sum_{i=1}^{\lambda} w_i^2$.
  $c_\sigma \leftarrow 3/K$. $\bar{c}_\sigma \leftarrow \sqrt{1 - (1 - c_\sigma))^2} \cdot \sqrt{\mu_{\vec{w}}}$.
  $c_\Sigma \leftarrow 4/K$. $\bar{c}_\Sigma \leftarrow \sqrt{1 - (1 - c_\Sigma))^2} \cdot \sqrt{\mu_{\vec{w}}}$.
  $c_1 \leftarrow 2/K^2$. $c_\mu \leftarrow \mu_{\vec{w}}/K^2$.
  $\mathbb{E} \leftarrow \sqrt{K} \cdot (1 - \frac{1}{4 \cdot K} - \frac{1}{21 \cdot K^2})$.
  $\vec{p}_\sigma \leftarrow \vec{0}$. $\vec{p}_\Sigma \leftarrow \vec{0}$.
  $t \leftarrow 0$.
  **while** $t < 1$ or $f(\vec{m}_t) - f(\vec{x}_{t-1}) > \epsilon$ **do**
    Sample $\vec{x}_1, \ldots, \vec{x}_\mu$ from the Gaussian distribution with mean $\vec{m}_t$ and covariance matrix $\sigma_t^2 \cdot \vec{\Sigma}_t$.
    Sort $\vec{x}_1, \ldots, \vec{x}_\mu$ ascendingly according to the objective function value $f(\vec{x}_i)$.
    $\vec{m}_{t+1} \leftarrow \sum_{i=1}^{\lambda} w_i \cdot \vec{x}_i$. ▷ Update Mean
    $\boldsymbol{V} \cdot \boldsymbol{\Lambda} \cdot \boldsymbol{V}^T \leftarrow \text{eig}(\boldsymbol{\Sigma}_t)$ ▷ eigenvalue decomposition
    $\vec{p}_\sigma \leftarrow (1 - c_\sigma) \cdot \vec{p}_\sigma + \bar{c}_\sigma \cdot \boldsymbol{V} \cdot \sqrt{\boldsymbol{\Lambda}}^{-1} \cdot \frac{\vec{m}_{t+1} - \vec{m}_t}{\sigma_t}$.
    $\sigma_{t+1} \leftarrow \sigma_t \cdot \exp(c_\sigma \cdot [\frac{\|\vec{p}_\sigma\|}{\mathbb{E}} - 1])$. ▷ Update $\sigma_t$.
    $\vec{p}_\Sigma \leftarrow (1 - c_\Sigma) \cdot \vec{p}_\Sigma$.
    **if** $\|\vec{p}_\sigma\| < 1.5 \cdot \sqrt{K}$ **then**
      $\vec{p}_\Sigma \leftarrow \vec{p}_\Sigma + \bar{c}_\Sigma \cdot \frac{\vec{m}_{t+1} - \vec{m}_t}{\sigma_t}$.
      $c_s \leftarrow 0$.
    **else**
      $c_s \leftarrow c_1 \cdot c_\Sigma \cdot (2 - c_\Sigma)$.
    **end if**
    $\boldsymbol{\Sigma}_{t+1} \leftarrow (1 - c_1 - c_\mu + c_s) \cdot \boldsymbol{\Sigma}_t + c_1 \cdot \vec{p}_\Sigma \cdot {p_\Sigma}^T + c_\mu \cdot \sum_{i=1}^{\lambda} w_i \cdot \frac{\vec{x}_i - \vec{m}_t}{\sigma_t} \cdot \frac{\vec{x}_i - \vec{m}_t}{\sigma_t}^T$. ▷ Update $\boldsymbol{\Sigma}_t$.
    $t \leftarrow t + 1$.
  **end while**
  **return** $\vec{x}_1$.
**end function**

procedure? In other words, how can we get as much information as possible from the few function evaluations that we do have?

Bayesian optimization addresses this challenge by optimizing not the objective function directly, but a surrogate model of the function which we obtain by means of *regression*. More precisely, Bayesian optimization approximates the objective function via a regression model with an uncertainty estimate and then chooses the next point such that it has a high chance to decrease our currently lowest objective function value. Acccordingly, we require two main ingredients for Bayesian optimization: First, a regression technique which provides an uncertainty estimate and second, a measure of sample utility if we have a regression model. We focus here on the most common regression technique for Bayesian optiization - Gaussian Processes - and the most common utility measures - the lower confidence bound and expected improvement. If you wish to get a deeper insight, you may want to refer to the works of Jones, Schonlau, and Welch (1998), Brochu, Cora, and Freitas (2010), and Frazier (2018).

Gaussian processes are a fascinating topic in their own right and we encourage interested students to have a deeper look into the topic (Rasmussen and Williams 2005, e.g.). For our intents and purposes, a Gaussian process model estimates the value of the objective function $f(\vec{x})$ at position $\vec{x} \in \mathbb{R}^K$ via a normal distribution with mean $\mu(\vec{x})$ and variance $\sigma^2(\vec{x})$. Roughly speaking, we compute the mean based on the objective function value of *similar* points that we have already sampled and the variance based on how many similar points there are. More specifically, assume that we have already sampled the points $(\vec{x}_1, y_1), \ldots, (\vec{x}_t, y_t)$ where $y_t \approx f(x_t)$. Further, we define a measure of similarity $k_\psi : \mathbb{R}^K \times \mathbb{R}^K \to \mathbb{R}$ as

$$k_\psi(\vec{x}, \vec{x}') = \exp\Big(-\frac{1}{2}\frac{\|\vec{x} - \vec{x}'\|^2}{\psi^2}\Big), \tag{2.33}$$

which we also call the *radial basis function kernel*. Note that $k_\psi$ is maximal if $\vec{x} = \vec{x}'$ and drops off in a Gaussian fashion if the distance between $\vec{x}$ and $\vec{x}'$ becomes larger. Based on this kernel function, we also define the auxiliary function $\vec{k} : \mathbb{R}^K \to \mathbb{R}^t$ and the auxiliary matrix $\boldsymbol{K} \in \mathbb{R}^{t \times t}$ where:

$$\vec{k}(\vec{x}) = \big(k(\vec{x}, \vec{x}_1), \ldots, k(\vec{x}, \vec{x}_t)\big)^T \qquad \text{and} \quad \boldsymbol{K}_{i,j} = k(\vec{x}_i, \vec{x}_j)$$

Finally, we define the auxiliary vector $\vec{y} = (y_1, \ldots, y_t)^T \in \mathbb{R}^t$.

Then, the Gaussian process estimate function $\mu : \mathbb{R}^K \to \mathbb{R}$ and the uncertainty function $\sigma^2 : \mathbb{R}^K \to \mathbb{R}$ are given as follows.

$$\mu(\vec{x}) = \vec{k}(\vec{x})^T \cdot \big(\boldsymbol{K} + \tilde{\sigma}^2 \cdot \boldsymbol{I}^t\big)^{-1} \cdot \vec{y} \tag{2.34}$$

$$\sigma^2(\vec{x}) = 1 - \vec{k}(\vec{x})^T \cdot \big(\boldsymbol{K} + \tilde{\sigma}^2 \cdot \boldsymbol{I}^t\big)^{-1} \cdot \vec{k}(\vec{x}) \tag{2.35}$$

where $\tilde{\sigma}$ is the assumed level of noise for our objective function estimates, typically set to small values such as $\tilde{\sigma} = 10^{-5}$.

Note that both of these functions can be evaluated efficiently because they only require a few matrix-vector multiplications after the matrix inversion is pre-computed once. Figure 2.13 (left) illustrates what our function estimate $\mu$ looks like for an example objective function and as few as six samples.

Once we have built a regression model like this, our next task is to select the best possible next sample, where "best possible" means a sample which may still improve our objective function value. The first approach to find such a value is to consider the standard

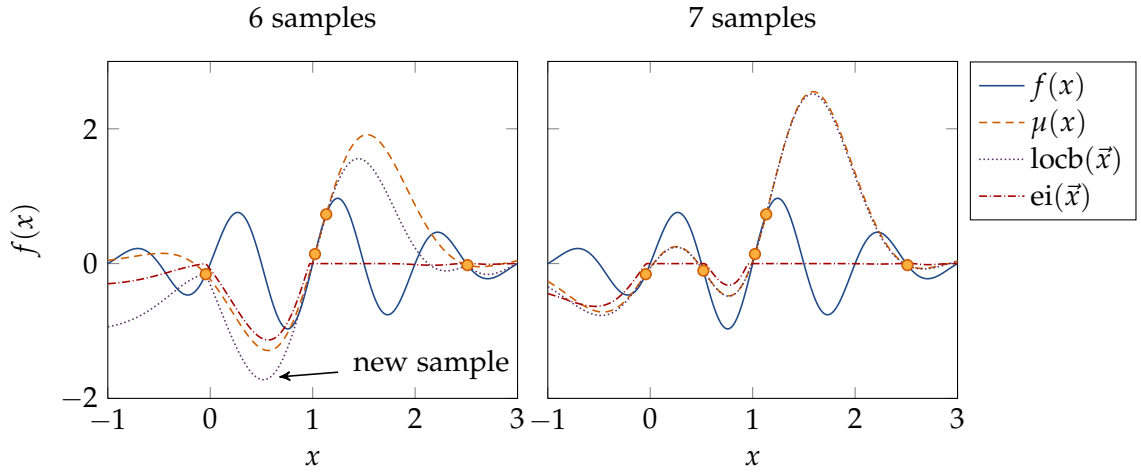



*Figure 2.13:* An illustration of Bayesian optimization for a single-dimensional optimization problem. Left: A Gaussian process approximation (shown in orange, fashed) of the true objective function $f$ (shown in blue), using the six samples shown as orange circles. We also show the lower confidence bound (purple, dotted) and the expected improvement curve (red, dot-dashed). Right: Another approximation for the same function after sampling one additional point (indicated by an arrow).

deviation of our Gaussian process regression and choose the value with the lowest *lower confidence bound* (locb), which is given as $\text{locb}(\vec{x}) = \mu(x) - \kappa \cdot \sigma(\vec{x})$, where $\kappa > 0$ is a hyper-parameter that expresses our desire to explore new values. It is recommended to reduce this value $\kappa$ over time (Brochu, Cora, and Freitas 2010). A second approach is to minimize the so-called *expected improvement* (ei), which is defined as follows.

$$\text{ei}(\vec{x}) := \int_{-\infty}^{f^*} p(y) \cdot [y - f^*]_- dy,$$

where $[x]_-$ is defined as $\min\{0, x\}$ and $f^*$ is the lowest objective function value we have achieved up to this points. In other words, we consider the expected amount by which $\vec{x}$ will reduce our currently best objective function value $f^*$, assuming that $p$ is a normal distribution with mean $\mu(\vec{x})$ and variance $\sigma^2(\vec{x})$. This integral can be solved in closed form and yields the following solution (Brochu, Cora, and Freitas 2010).

$$\text{ei}(\vec{x}) = \big(\mu(\vec{x}) - f^*\big) \cdot P\big(z(\vec{x})\big) - \sigma(\vec{x}) \cdot p\big(z(\vec{x})\big) \quad \text{where} \quad z(\vec{x}) := \frac{f^* - \mu(\vec{x})}{\sigma(\vec{x})},$$

where $p$ is the density function of the standard normal distribution and where $P$ is the cumulative density function of the standard normal distribution.

Note again that both locb and ei can be computed efficiently and thus also optimized efficiently using other means. It is even possible to compute gradients of those functions (although we omit this here) and thus apply gradient-based methods for optimization. Figure 2.13 illustrates the lower confidence bound and the expected improvement for a simple, one-dimensional problem. As next sample, we choose the point which minimizes the lower confidence bound and thus obtain the new model on the right.

The overall Bayesian Optimization algorithm is illustrated in Algorithm 20.

**Algorithm 20** Bayesian optimization for an objective function $f : \mathbb{R}^K \to \mathbb{R}$, initial samples $(\vec{x}_1, y_1), \ldots, (\vec{x}_t, y_t)$ where $y_i \approx f(\vec{x}_i)$ for all $i \in \{1, \ldots, t\}$, an assumed noise variance $\tilde{\sigma}^2 > 0$, a kernel bandwidth $\psi > 0$, and a maximum number of function evaluations $T$.

**function** BAYES-OPT(objective function $f : \mathbb{R}^K \to \mathbb{R}$, initial samples $(\vec{x}_1, y_1), \ldots, (\vec{x}_t, y_t) \in \mathbb{R}^K \times \mathbb{R}$, noise variance $\tilde{\sigma}^2 > 0$, kernel bandwidth $\psi > 0$, budget $T$.)
  **while** $t < T$ **do**
    $\vec{y} \leftarrow (y_1, \ldots, y_t)^T$.
    Compute $\boldsymbol{K} \in \mathbb{R}^{t \times t}$ with $\boldsymbol{K}_{i,j} = \exp\left( -\frac{1}{2} \frac{\|\vec{x}_i - \vec{x}_j\|^2}{\psi^2} \right)$.
    Compute $\left( \boldsymbol{K} + \tilde{\sigma}^2 \cdot \boldsymbol{I}^t \right)^{-1}$.
    Define $\vec{k} : \mathbb{R}^K \to \mathbb{R}$ via $k_i(\vec{x}) = \exp\left( -\frac{1}{2} \frac{\|\vec{x} - \vec{x}_i\|^2}{\psi^2} \right)$.
    Define $\mu : \mathbb{R}^K \to \mathbb{R}$ as in Equation 2.34
    Define $\sigma : \mathbb{R}^K \to \mathbb{R}$ as in Equation 2.35
    Select a utility function $u : \mathbb{R}^K \to \mathbb{R}$, either locb or ei.
    $\vec{x}_{t+1} \leftarrow \operatorname{argmin}_{\vec{x}} u(\vec{x})$.
    $y_{t+1} \leftarrow f(\vec{x})$.
    $t \leftarrow t + 1$.
  **end while**
  $t^* \leftarrow \operatorname{argmin}_t y_t$.
  **return** $\vec{x}_{t^*}, y_{t^*}$.
**end function**

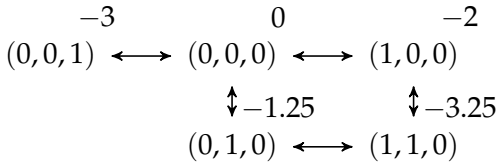



*Figure 2.14:* The feasible set of the knapsack problem from Example 87. The bit-flip neighborhood is indicated by arrows. The objective function value is indicated by the number above each binary vector.

### 2.5.2 *Discrete Optimization*

Discrete Optimization deals with optimization problems where the domain is *countable*, in the sense that there exists a surjective map from the natural numbers to the domain (also refer to Definition 22). Examples of such countable domains are the natural numbers themselves, but also integers, binary vectors, strings, graphs, and many more.

The difficulty of discrete optimization problems lies in the fact that we can not move smoothly through the domain, which we implicitly or explicitly did in all our previous methods. Instead, we need to make "jumps" from one point to the next. In many cases, this implies that the problems are *NP-hard*, which is to say that there exists no known algorithm which can solve them exactly in less than exponential time. Due to these difficulties, we rely on heuristics, the most popular of which we cover in this section.

**Example 87** (Knapsack Problem)**.** Consider the problem of packing optimally for a trip. You could pack anything in a set of $K$ objects, each of which has a utility $u_k$ and a weight $w_k$. However, you can carry at most $\hat{w}$ kilograms of weight. The problem of maximizing the utility of the stuff you carry while keeping within the weight limit is then given as follows.

$$\begin{aligned} \min_{\vec{x} \in \mathbb{Z}^K} \quad & -\vec{u}^T \cdot \vec{x} \\ \text{s.t.} \quad & \vec{w}^T \cdot \vec{x} \leqslant \hat{w} \\ & \vec{0} \leqslant \vec{x} \leqslant \vec{1} \end{aligned}$$

Such a problem is, in general, NP-hard and is only feasible to solve exactly for small instances.

For example, consider the following three-item problem with the utilities $\vec{u} = (2, 1.25, 3)^T$, the weights $\vec{w} = (3, 2, 4)^T$, and the weight limit $\hat{w} = 5$. The feasible set of the problem and the respective objective function values are shown in Figure 2.14. Accordingly, the global minimum is $x = (1, 1, 0)^T$ with objective function value $f(x) = -3.25$.

#### Hill Climbing

The simplest approach to perform discrete optimization is to start from some point $x_0 \in \mathcal{X}^*$ and then move to a neighboring point that decreases the objective function value the most. The resulting algorithm is shown in Algorithm 21.

Note that this algorithm relies on a neighborhood function, which may be problem-specific and needs to be efficiently computable. In case the neighborhood is too large for a problem, it is also possible to consider only a randomly sampled subset of the

**Algorithm 21** Hill climbing for an objective function $f : \mathcal{X} \to \mathbb{R}$, an initial point $x_0 \in \mathcal{X}$, a neighborhood function $\mathcal{N} : \mathcal{X} \to \mathcal{P}(\mathcal{X})$, and a feasible set $\mathcal{X}^*$.

```
function HILL-CLIMB(objective function f : X → ℝ, initial point x_0 ∈ X, neighborhood function N : X → P(X), feasible set X*)
    t ← 0.
    while t = 0 or f(x_t) < f(x_{t−1}) do
        x_{t+1} ← argmin_{x∈N(x_t)∩X*} f(x).
        t ← t + 1.
    end while
    return x_{t−1}.
end function
```

neighborhood. Also note that we assume that all constraint functions can be evaluated quickly in order to restrict the considered neighborhood to the feasible set.

Another challenge lies in suboptimal local minima because hill climbing can intrinsically not escape such local minima. As an example, consider the knapsack example 87 from before. If we start from $x_0 = (0, 0, 0)$, hill climbing would move to $x_1 = (0, 0, 1)$ with objective function value $f(x_1) = -3$ and then get stuck.

More generally speaking, hill climbing puts too much emphasis on *exploitation* compared to *exploration*, i.e. we ensure that we find the best value in our immediate neighborhood but do not explore enough of the entire feasible set to find the best neighborhood. We therefore require a strategy which strikes a better compromise between exploration and exploitation.

### Simulated Annealing

Simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983) is inspired by statistical mechanics where systems settle in favourable energy states if they are first heated and then slowly cooled. This occurs because a system fluctuates a lot at high temperature, such that many possible configurations are explored in a short time period. While cooling, the fluctuation reduces and the system is more likely to settle in a closeby state with lower energy, i.e. exploitation is emphasized more over time.

In more detail, energy configurations at temperature $T$ are distributed according to the *Boltzmann distribution*, which assigns a probability proportional to $\exp(-\alpha \cdot f/T)$ to a state with energy $f$, where $\alpha$ is some conversion constant between temperature and energy.

To transfer these ideas to optimization, we assign a probability $P(x)$ to all neighbors of $x_t$ as follows.

$$P(x) = \frac{w(x)}{\sum_{x \in \mathcal{N}(x_t)} w(x)} \quad \text{where} \quad w(x) = \begin{cases} 1 & \text{if } f(x) \leqslant f(x_t) \\ \exp\left[-\frac{f(x)-f(x_t)}{T}\right] & \text{otherwise} \end{cases} \tag{2.36}$$

In other words, we move to all neighbors which improve the objective function value with equal probability and move to all neighbors which would make the objective function value worse with a probability according to the Boltzmann distribution. The temperature $T$ is then reduced over time. The resulting algorithm is shown in Algorithm 22.

### Tabu Search

Tabu search (Glover 1986) is another variant of hill climbing to avoid getting stuck in local minima. The first change to hill climbing is to permit to move to worse points if no

**Algorithm 22** Simulated Annealing for an objective function $f : \mathcal{X} \to \mathbb{R}$, an initial point $x_0 \in \mathcal{X}$, a neighborhood function $\mathcal{N} : \mathcal{X} \to \mathcal{P}(\mathcal{X})$, a feasible set $\mathcal{X}^*$, an initial temperature $T_0 > 0$, a cooling factor $\alpha \in (0, 1)$, and a minimum temperature $0 < T_{\min} < T_0$.

**function** SIMULATED-ANNEALING(objective function $f : \mathcal{X} \to \mathbb{R}$, initial point $x_0 \in \mathcal{X}$, neighborhood function $\mathcal{N} : \mathcal{X} \to \mathcal{P}(\mathcal{X})$, feasible set $\mathcal{X}^*$, initial temperature $T_0 > 0$, cooling factor $\alpha \in (0, 1)$, minimum temperature $0 < T_{\min} < T_0$.)
  $t \leftarrow 0$. $T \leftarrow T_0$.
  **while** $T > T_{\min}$ or $f(x_t) < f(x_{t-1})$ **do**
    $y_1, \dots, y_m \leftarrow \mathcal{N}(x_t) \cap \mathcal{X}^*$.
    **for** $i \in \{1, \dots, m\}$ **do**
      $w_i \leftarrow \max\{1, \exp\big[-\frac{f(y_i)-f(x_t)}{T}\big]\}$. ▷ Boltzmann distribution
    **end for**
    **for** $i \in \{1, \dots, m\}$ **do**
      $p_i \leftarrow w_i / \big(\sum_{i=1}^{m} w_i\big)$.
    **end for**
    Sample $i$ randomly according to probabilities $p_i$.
    $x_{t+1} \leftarrow y_i$.
    $T \leftarrow T \cdot \alpha$. ▷ anneal temperature
    $t \leftarrow t + 1$.
  **end while**
  **return** $x_{t-1}$.
**end function**

better option is available in the neighborhood. The second change is to forbid moving to points that have already been visited in the last $T$ steps (these points are 'taboo'). This way, tabu search prevents getting stuck in regions of the search space with an isolated local minimum that has only a narrow basin of attraction.

Tabu search poses two implementation challenges. First, there is no immediately obvious condition to stop the search because we could always keep moving as long as our non-taboo neighborhood is not empty. Therefore, we need some new kind of stopping criterion. In this case, we go for a certain number of moves $\tau$, after which only accept improvements in the objective function value.

Another challenge is that the tabu list may forbid the entire neighborhood of a point such that we get stuck. Such a case can be avoided by making the oldest element in the tabu list available again. The resulting algorithm is shown in Algorithm 23.

This concludes our short overview of local search approaches for discrete optimization. However, it is also possible to perform discrete optimization - at least for certain cases - without local search. Two famous options are branch & cut as well as ant colony optimization, which we will cover next.

### Branch and Cut

*Branch and cut* (Padberg and Rinaldi 1991) is an algorithm to solve *integer linear programs (ILPs)*. An *ILP* is a variation of the linear program in Equation 2.30, where we restrict the domain to contain only integers. In other words, we consider problems of the following form.

$$\begin{aligned} \min_{\vec{x} \in \mathbb{Z}^K} \quad & \vec{c}^T \cdot \vec{x} \\ \text{s.t.} \quad & \boldsymbol{A} \cdot \vec{x} \leqslant \vec{b} \end{aligned} \tag{2.37}$$

**Algorithm 23** Tabu search for an objective function $f : \mathcal{X} \to \mathbb{R}$, an initial point $x_0 \in \mathcal{X}$, a neighborhood function $\mathcal{N} : \mathcal{X} \to \mathcal{P}(\mathcal{X}^*)$, a tabu list length $T \in N$, and a minimum number of moves $\tau \in \mathbb{N}$.

1: **function** TABU-SEARCH(objective function $f : \mathcal{X} \to \mathbb{R}$, initial point $x_0 \in \mathcal{X}$, neighborhood function $\mathcal{N} : \mathcal{X} \to \mathcal{P}(\mathcal{X}^*)$, tabu list length $T \in N$,minimum number of moves $\tau \in \mathbb{N}$.)
2: $t \leftarrow 0$.
3: $\Phi \leftarrow \varnothing$.
4: **while** $t < \tau$ or $f(x_t) < f(x_{t-1})$ **do**
5: **if** $\mathcal{N}(x_t) \backslash \Phi \neq \varnothing$ **then**
6: $x_{t+1} \leftarrow \text{argmin}_{x \in \mathcal{N}(x_t) \backslash \Phi} f(x)$.
7: **else**
8: $x_{t+1} \leftarrow$ oldest element in $\Phi$.
9: **end if**
10: $\Phi \leftarrow \Phi \cup \{x_t\}$. ▷ $x_t$ is now taboo
11: **if** $|\Phi| > \tau$ **then**
12: Remove oldest element from $\Phi$.
13: **end if**
14: $t \leftarrow t + 1$.
15: **end while**
16: **return** $x_{t-1}$.
17: **end function**

$$A^{\text{eq}} \cdot \vec{x} = \vec{b}^{\text{eq}}$$
$$\vec{lb} \leqslant \vec{x} \leqslant \vec{ub},$$

Just as linear programs, ILPs occur quite frequently in computer science. Due to their discrete nature, however, there are much harder to solve. More specifically, they are NP-hard.

While we could employ local search heuristics as before, the intuitively most straightforward heuristic for an ILP is to solve instead the continuous linear program - which is possible efficiently - and then round all values of the solution vector to the nearest integer. Unfortunately, though, the rounding may worsen the objective function and, even worse, violate side constraints.

A smarter alternative is offered by the *branch and cut* approach. In this method, we first solve the continuous version of the ILP and if our solution vector is not yet an integer vector, we construct two alternative linear programs for a non-integer entry $x_k$ of our vector $\vec{x}$, namely one program where $x_k$ is upper-bounded by $\lfloor x_k \rfloor$, and one program where $x_k$ is lower-bounded by $\lceil x_k \rceil$. Then we iterate the procedure. This yields Algorithm 24. Figure 2.15 shows the algorithm applied to the knapsack example 87.

Note that branch-and-cut may need many branches to arrive at the optimal solution. Therefore, using a reasonable strategy to decide on the best branch is key. However, a nice property of branch-and-cut is that it requires no hyper-parameter choices, not even an initial point, which makes it easier to apply.

### Ant Colony Optimization

*Ant colony optimization (ACO)* (Dorigo and Di Caro 1999) is an optimization technique for graph problems which is inspired by biological research. In particular, it has been

**Algorithm 24** Branch-and-Cut algorithm for an ILP with parameters $\vec{c}$, $\boldsymbol{A}$, $\vec{b}$, $\boldsymbol{A}^{\text{eq}}$, $\vec{b}^{\text{eq}}$, $\vec{lb}$, and $\vec{ub}$, assuming a solver for linear programs LP.

**function** BRANCH-AND-CUT(parameters $\vec{c}$, $\boldsymbol{A}$, $\vec{b}$, $\boldsymbol{A}^{\text{eq}}$, $\vec{b}^{\text{eq}}$, $\vec{lb}$, and $\vec{ub}$.)
  $\Phi \leftarrow \{(\vec{lb}, \vec{ub})\}$.
  $f^* \leftarrow \infty$. $\vec{x}^* \leftarrow \vec{0}$.
  **while** $\Phi \neq \varnothing$ **do**
    Poll bounds $(\vec{lb}, \vec{ub})$ from $\Phi$.
    $\vec{x} \leftarrow \text{LP}(\vec{c}, \boldsymbol{A}, \vec{b}, \boldsymbol{A}^{\text{eq}}, \vec{b}^{\text{eq}}, \vec{lb}, \vec{ub})$ ▷ solve relaxed problem
    **if** $\vec{c}^T \cdot \vec{x} \geqslant f^*$ **then**
      **Continue**.
    **end if**
    **if** $\vec{x} \in \mathbb{Z}^K$ **then**
      $f^* \leftarrow \vec{c}^T \cdot \vec{x}$. $\vec{x}^* \leftarrow \vec{x}$.
      **Continue**.
    **end if**
    Let $k \leftarrow \text{argmax}_{k \in \{1,\dots,K\}} |x_k - \text{round}(\vec{x}_k)|$. ▷ Select non-integer variable
    $\tilde{lb} \leftarrow \vec{lb}$. $\tilde{lb}_k \leftarrow \lceil x_k \rceil$. ▷ Cut
    $\tilde{ub} \leftarrow \vec{ub}$. $\tilde{ub}_k \leftarrow \lfloor x_k \rfloor$. ▷ Cut
    $\Phi \leftarrow \Phi \cup \{(\vec{lb}, \tilde{ub}), (\tilde{lb}, \vec{ub})\}$. ▷ Branch
  **end while**
  **return** $\vec{x}^*, f^*$.
**end function**

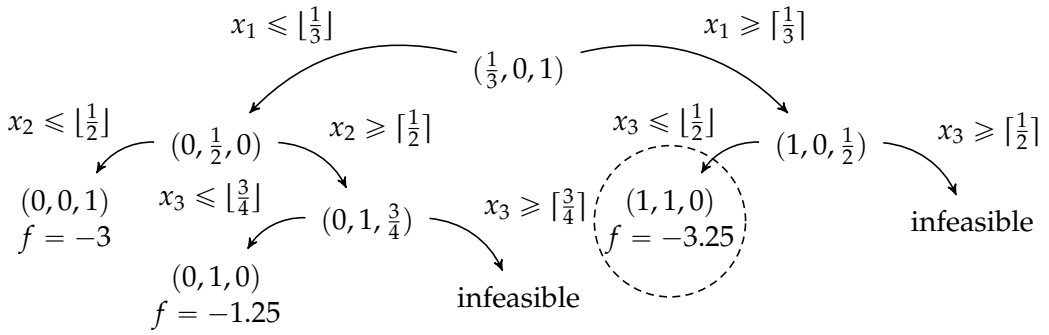



*Figure 2.15:* An illustration of the branch-and-cut Algorithm 24 applied to Example 87. We start off with the solution to the continuous version of the integer linear program at the top and then solve two new linear programs where the value of $x_1$ is restricted to $\leqslant 0$ and $\geqslant 1$ respectively. This yields new solutions for which we repeat the procedure until we achieve an integer solution (here: $\vec{x} = (0, 0, 1)$). Then we store the respective objective function value (here: $f(\vec{x}) = -3$) and prune all subsequent computations with a worse value. The best solution is highlighted with a dashed circle.

observed that groups of Argentinian ants are quite efficient in finding shortest paths to a food source even though a single ant would not have the capacity to perform such an optimization. A model that explains this behavior is as follows. First, every ant moves at random, starting from the nest, until it finds a food source. Then it tracks back its path and leaves a pheromone trail on it. Subsequent ants will now be biased to follow the pheromone trail but may still break off and discover faster routes to the same food source. Because ants that discover a shorter path will be quicker in leaving a pheromone trail, such paths will be reinforced until the shortest path contains so much pheromone that all ants move along this path.

To exploit such behavior in an algorithm, we require three ingredients. First, a function $\psi$ which constructs a feasible path of an ant through a given graph, based on the pheromone that is already present. Second, an objective function $f$ for the path. And third, a function $\phi$ which controls the pheromone update along a path, given its objective function value. The abstract pseudocode for ACO is shown in Algorithm 25.

**Algorithm 25** Ant colony optimization (ACO) for a path construction function $\psi : \mathbb{R}^{m \times m} \to \Pi(\{1, \dots, m\})$, an objective function $f : \Pi(\{1, \dots, m\}) \to \mathbb{R}$, a pheromone update function $\phi : \mathbb{R} \to \mathbb{R}$, a number of iterations $T \in \mathbb{N}$, and a pheromone evaporation rate $\alpha \in [0, 1]$, where $\Pi(\{1, \dots, m\})$ is the set of all permutations of $\{1, \dots, m\}$, i.e. the set of all possible paths in the graph.

**function** ACO(path function $\psi$, objective function $f$, update function $\phi$, number of iterations $T \in \mathbb{N}$, evaporation rate $\alpha \in [0, 1]$)
  Initialize an initial pheromone matrix $\mathbf{\Phi} \in \mathbb{R}^{m \times m}$.
  $c^* \leftarrow \infty$.
  **for** $t \in \{1, \dots, T\}$ **do**
    $(i_1, \dots, i_n) \leftarrow \psi(\mathbf{\Phi})$. ▷ Compute path
    $c \leftarrow f(i_1, \dots, i_n)$. ▷ Compute value of path
    **if** $c < c^*$ **then**
      $\pi^* \leftarrow (i_1, \dots, i_n)$. $c^* \leftarrow c$.
    **end if**
    **for** $j \in \{1, \dots, n-1\}$ **do**
      $\Phi_{i_j, i_{j+1}} \leftarrow \Phi_{i_j, i_{j+1}} + \phi(c)$. ▷ Update pheromone
    **end for**
    **for** $i \in \{1, \dots, m\}$ **do**
      **for** $h \in \{1, \dots, m\}$ **do**
        $\Phi_{i,j} \leftarrow \Phi_{i,j} \cdot (1 - \alpha)$. ▷ Evaporate pheromone
      **end for**
    **end for**
  **end for**
  **return** $\pi^*$.
**end function**

To make ACO applicable in practice, we need to decide on useful forms of $\psi$, $f$, and $\phi$ for our problem in question. The most straightforward application is the traveling salesperson problem from Example 5. In the TSP, we are looking for a shortest round trip between $m$ cities, for which the pairwise distances are recorded in the matrix $\boldsymbol{D} \in \mathbb{R}^{m \times m}$. Accordingly, the functions $\psi$, $f$, and $\phi$ can be chosen as specified in Algorithm 26. A visualization is shown in Figure 2.16.

**Algorithm 26** The path construction function $\psi$, the objective function $f$, and the update function $\phi$ for ant colony optimization (ACO) for the traveling salesperson problem with pairwise distance matrix $\boldsymbol{D} \in \mathbb{R}^{m \times m}$.

**function** $\psi$(Pheromone matrix $\boldsymbol{\Phi} \in \mathbb{R}^{m \times m}$)
  $t \leftarrow 1$.
  $i_t \leftarrow 1$.
  $V \leftarrow \{2, \ldots, m\}$. ▷ Remaining nodes.
  **for** $t \leftarrow \{2, \ldots, m-1\}$ **do**
    $\vec{p} \leftarrow \vec{0} \in \mathbb{R}^m$.
    **for** $j \in V$ **do**
      $p_j \leftarrow \Phi_{i_t,j} / D_{i_t,j}$.
    **end for**
    $\vec{p} \leftarrow \vec{p} / \left( \sum_{j=1}^{m} p_j \right)$.
    Sample $j$ randomly with probability $p_j$.
    $i_{t+1} \leftarrow j$.
    $V \leftarrow V \backslash \{j\}$.
  **end for**
  **return** $i_1, \ldots, i_m$.
**end function**
**function** $f$(Path $i_1, \ldots, i_m$)
  **return** $D_{i_1,i_2} + \ldots + D_{i_{m-1},i_m} + D_{i_m,i_1}$.
**end function**
**function** $\phi$(objective function value $c$)
  **return** $1/c$.
**end function**

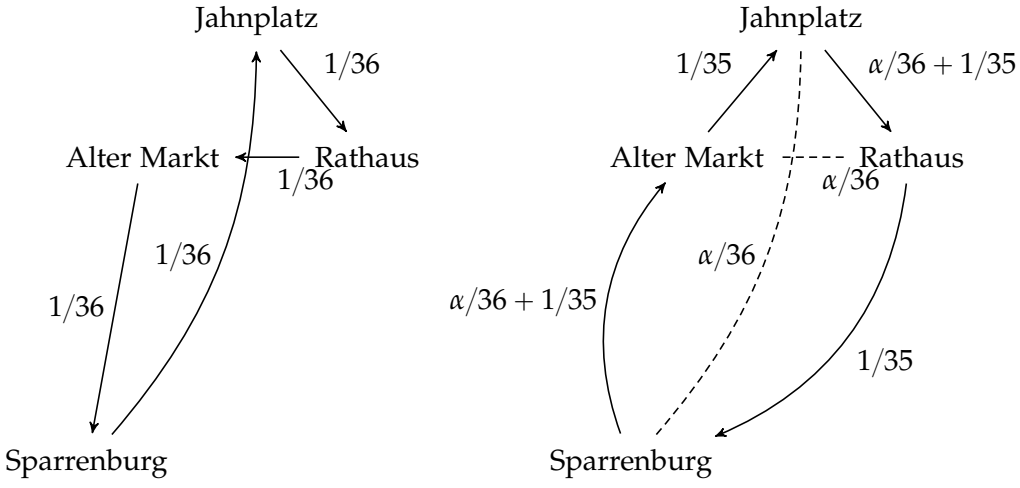



*Figure 2.16:* A visualization of two iterations of the traveling salesperson version of ant colony optimization (ACO). The solid lines show the path of the current ant and the numbers display the pheromone value right after the pheromone update step. The pheromone put on the path grows for shorter paths. Further, edges that are used by multiple ants accumulate more pheromone while the pheromone on edges that ae used less often evaporates exponentially over time.

# BIBLIOGRAPHY


Barber, David (2010). *Bayesian Reasoning and Machine Learning*. Cambridge, UK: Cambridge University Press. URL: www.cs.ucl.ac.uk/staff/D.Barber/brml.

Boyd, Stephen and Lieven Vandenberghe (2004). *Convex Optimization*. Cambridge, UK: Cambridge University Press. ISBN: 9780521833783. URL: http://web.stanford.edu/~boyd/cvxbook/.

Brochu, Eric, Vlad Cora, and Nando de Freitas (2010). *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*. arXiv: 1012.2599 [cs.LG].

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society. Series B* 39.1, pp. 1–38. URL: https://www.jstor.org/stable/2984875.

Dorigo, M. and G. Di Caro (1999). "Ant colony optimization: a new meta-heuristic". In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*. Vol. 2, pp. 1470–1477. DOI: 10.1109/CEC.1999.782657.

Fackler, Paul L. (2005). *Notes on Matrix Calculus*. North Carolina State University. URL: http://www2.stat.duke.edu/~zo2/shared/resources/matrixc1.pdf.

Frazier, Peter I. (2018). *A Tutorial on Bayesian Optimization*. arXiv: 1807.02811 [stat.ML].

Glover, Fred (1986). "Future paths for integer programming and links to artificial intelligence". In: *Computers & Operations Research* 13.5. Applications of Integer Programming, pp. 533–549. DOI: 10.1016/0305-0548(86)90048-1.

Hillier, Frederick S. and Gerald J. Liebermann (2010). *Introduction to Operations Research*. 9th ed. New York City, NY, USA: McGraw-Hill. ISBN: 9780073376295.

Jones, Donald R., Matthias Schonlau, and William J. Welch (1998). "Efficient Global Optimization of Expensive Black-Box Functions". In: *Journal of Global Optimization* 13.4, pp. 455–492. DOI: 10.1023/A:1008306431147.

Kingma, Diederik P. and Jimmy Lei Ba (2015). "Adam: A Method for Stochastic Optimization". In: *Proceedings of the Third International Conference on Learning Representations (ICLR 2015)*. Ed. by Yoshua Bengio et al. URL: https://arxiv.org/abs/1412.6980.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). "Optimization by Simulated Annealing". In: *Science* 220.4598, pp. 671–680. DOI: 10.1126/science.220.4598.671.

Nelder, J. A. and R. Mead (1965). "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4, pp. 308–313. DOI: 10.1093/comjnl/7.4.308.

Nocedal, Jorge and Stephen J. Wright (1999). *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. New York: Springer. DOI: 10.1007/b98874.

Padberg, M. and G. Rinaldi (1991). "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems". In: *SIAM Review* 33.1, pp. 60–100. DOI: 10.1137/1033004.

Rasmussen, Carl Edward and Christopher K. I. Williams (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. Cambridge, MA, USA: The MIT Press.

Simon, Dan (2013). *Evolutionary Optimization Algorithms*. Hoboken, NJ, USA: Wiley. ISBN: 978-0-470-93741-9.

Wolfe, Philip (1961). "A duality theorem for non-linear programming". In: *Quarterly of Applied Mathematics* 19.3, pp. 239–244. DOI: 10.1090/qam/135625.

# ACRONYMS

**ACO** ant colony optimization. 91, 93, 94
**Adam** adaptive moment estimation. 46, 47

**BFGS** Broyden-Fletcher-Goldfarb-Shanno algorithm. 51, 52, 54, 55

**CMA-ES** covariance matrix adaptation evolutionary strategy. 82–84

**EM** Expectation Maximization. 67, 70, 73, 74

**GMM** Gaussian Mixture Model. 68, 70, 73, 74

**ILP** integer linear program. 90–92

**KKT** Karush-Kuhn-Tucker condition. 28, 55

# GLOSSARY

**convex** A *set* is called convex if the connecting line between any two points in the set is also in the set; a *function* is called convex if the connecting line between any two points on the curve is above the curve; an optimization problem is called convex if both the objective function and the feasible set are convex; refer to Definition 37. 17–23, 27, 28, 31, 32, 35–40, 42, 43, 45, 49, 58, 62, 63, 66, 71–73, 77, *see* objective function & feasible set

**domain** ($\mathcal{X}$) The set of values for the variable in an optimization problem; refer to Definition 1. 1, 2, 5, 7, 8, 10, *see* variable

**equality constraint** ($h_j(\vec{x}) = 0$) an equation that limits the possible values the variable of an optimization problem; refer to Definition 1. 1, 2, 5, 21, 22, 29, 31, 38–41, 60–64, 79, *see* variable

**feasible set** ($\mathcal{X}^*$) The subset of the domain for which all inequality and equality constraints are fulfilled. An element of the feasible set is called *feasible point*; refer to Definition 11. 5, 7–10, 17, 21, 22, 26, 29, 36, 58, 60–62, 64, 88–90, *see* domain, equality constraint & inequality constraint

**global minimum** ($x^*$) An element $x^*$ from the feasible set $\mathcal{X}^*$ of a minimization problem, such that for all $x \in \mathcal{X}^*$ it holds $f(x^*) \leqslant f(x)$, where $f$ is the objective function; refer to Definition 11. 7–9, 15–17, 19, 29, 31–33, 35–40, 42, 43, 45, 51, 55, 63, 66, 67, 70–73, 77, 88, *see* minimization problem, feasible set & objective function

**inequality constraint** ($g_i(\vec{x}) \geqslant 0$) an inequality that limits the possible values the variable of an optimization problem; refer to Definition 1. 1, 2, 5, 21, 22, 27–29, 31, 41, 58–60, 78, *see* variable

**Karush-Kuhn-Tucker condition** A list of necessary conditions for the optimum of a constrained optimization problem, using the Lagrange dual; refer to Definition 55. 28–31, 97, *see* Lagrange dual

**Lagrange dual** ($\mathcal{L}$) The Lagrange dual of a problem is an alternative version of a problem where inequality and equality constraints are incorporated as terms in the objective function. $\mathcal{L}$ denotes this alternative objective function; refer to Definition 45. 24–33, 38, 40, 55, 62, 72, *see* inequality constraint & equality constraint

**local minimum** ($x^*$) An element $x^*$ from the feasible set $\mathcal{X}^*$ of a minimization problem, such that for all $x \in \mathcal{X}^*$ in a neighborhood around $x^*$ it holds $f(x^*) \leqslant f(x)$, where $f$ is the objective function; refer to Definition 15. 8, 9, 12–17, 19, 25, 28, 29, 35–39, 42, 43, 45, 51, 52, 58–60, 77, 80, 89, 90, *see* minimization problem, feasible set & objective function

**maximization problem** A problem given by a domain, an objective function, a list of equality constraints, and a list of inequality constraints. We wish to find an element from the feasible set which maximizes the output of the objective function; refer to Definition 1. 1, 6, *see* domain, objective function, inequality constraint, equality constraint & feasible set

**minimization problem** A problem given by a domain, an objective function, a list of equality constraints, and a list of inequality constraints. We wish to find an element from the feasible set which minimizes the output of the objective function; refer to

Definition 1. 1, 2, 5, 6, *see* domain, objective function, inequality constraint, equality constraint & feasible set

**objective function** ($f$) a function that maps from the domain to the real numbers. In a minimization problem, we wish to find the input value which minimizes the output of the objective function; refer to Definition 1. 1, 2, 5–11, 16, 17, 22, 26, 27, 29, 32, 33, 35–40, 42–52, 54–62, 64, 79–94, *see* domain

**optimization problem** Either a minimization problem or a maximization problem; can be re-written into a standard minimization form; refer to Definition 1. 1, 2, 6–8, 10, 13, 14, 17, 19, 21, 22, 24, 26–29, 31–33, 35, 38, 43, 44, 46, 50, 55–57, 62, 65, 67, 70, 77, 88, *see* minimization problem & maximization problem

**positive definite** A symmetric square matrix $\boldsymbol{A}$ is called positive definite if for all nonzero $\vec{x}$ it holds $\vec{x}^T \cdot \boldsymbol{A} \cdot \vec{x} > 0$; equivalent to having only positive eigenvalues; refer to Definition 29 and Theorem 35. 12–16, 28, 36–40, 51, 55, 68, 71, 72

**positive semi-definite** A symmetric square matrix $\boldsymbol{A}$ is called positive definite if for all $\vec{x}$ it holds $\vec{x}^T \cdot \boldsymbol{A} \cdot \vec{x} \geqslant 0$; equivalent to having only non-negative eigenvalues; refer to Definition 29 and Theorem 35. 12–16, 18, 19, 21, 35, 37, 78

**Slater's condition** Slater's condition requires that an optimization problem is convex, that all equality constraints are affine, and that at least one point exists for which all inequality constraints are strictly fulfilled; this constraint guarantees strong duality; refer to Definition 53 and Theorem 54. 27–29, 31, 33, 38–40, *see* convex, equality constraint & inequality constraint

**variable** ($\vec{x}$) Also: *parameter*; a placeholder for the values we can change/vary in order to minimize or maximize the objective function in an optimization problem; refer to Definition 1. *see* domain & objective function

**Wolfe dual** The Wolfe dual of a problem is a version of the Lagrange dual where we already plug in the Karush-Kuhn-Tucker conditions; refer to Definition 59. 31–33, 79, *see* Lagrange dual & Karush-Kuhn-Tucker condition

# RULES FOR DERIVATIVES AND GRADIENTS

## GENERAL RULES

Let $\vec{x} \in \mathbb{R}^m$, let $f : \mathbb{R}^m \to \mathbb{R}$, let $g : \mathbb{R}^m \to \mathbb{R}$, let $h : \mathbb{R} \to \mathbb{R}$, and let $\alpha, \beta \in \mathbb{R}$ be constants.

$$\nabla_{\vec{x}}\big(\alpha \cdot f(\vec{x}) + \beta \cdot g(\vec{x})\big) = \alpha \cdot \nabla_{\vec{x}} f(x) + \beta \cdot \nabla_{\vec{x}} g(x) \qquad \text{(linearity rule)}$$

$$\nabla_{\vec{x}}\big(f(\vec{x}) \cdot g(\vec{x})\big) = \big(\nabla_{\vec{x}} f(\vec{x})\big) \cdot g(\vec{x}) + f(\vec{x}) \cdot \big(\nabla_{\vec{x}} g(\vec{x})\big) \qquad \text{(product rule)}$$

$$\nabla_{\vec{x}} \frac{f(\vec{x})}{g(\vec{x})} = \frac{\big(\nabla_{\vec{x}} f(\vec{x})\big) \cdot g(\vec{x}) - f(\vec{x}) \cdot \big(\nabla_{\vec{x}} g(\vec{x})\big)}{g(\vec{x})^2} \qquad \text{(quotient rule)}$$

$$\nabla_{\vec{x}} h(g(\vec{x})) = \big(\frac{\partial}{\partial g(\vec{x})} h(g(\vec{x}))\big) \cdot \nabla_{\vec{x}} g(\vec{x}) \qquad \text{(chain rule)}$$

## SCALAR DERIVATIVES

Let $c, n$ be constants.

$$\frac{\partial}{\partial x} c \cdot x = c, \qquad \frac{\partial}{\partial x} x^n = n \cdot x^{n-1}, \qquad \frac{\partial}{\partial x} \frac{1}{x^n} = -n \cdot \frac{1}{x^{n+1}} \qquad \text{(polynom rules)}$$

$$\frac{\partial}{\partial x} \exp(x) = \exp(x), \qquad \frac{\partial}{\partial x} \log(x) = \frac{1}{x} \qquad \text{(exponential / log rule)}$$

## MATRIX CALCULUS

Let $\vec{x} \in \mathbb{R}^m$, let $\boldsymbol{A} \in \mathbb{R}^{m \times m}$, and let $\vec{y} \in \mathbb{R}^m$.

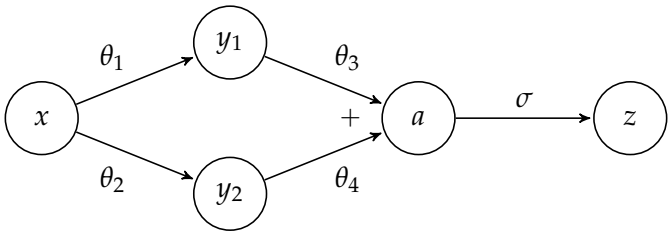
$$\begin{aligned}
\nabla_{\vec{x}} \vec{y} &= 0 \\
\nabla_{\vec{x}} \vec{y}^T \cdot \vec{x} &= \nabla_{\vec{x}} \vec{x}^T \cdot \vec{y} = \vec{y} \\
\nabla_{\vec{x}} \vec{x}^T \cdot \boldsymbol{A} \cdot \vec{x} &= (\boldsymbol{A}^T + \boldsymbol{A}) \cdot \vec{x} \\
\nabla^2_{\vec{x}} \vec{x}^T \cdot \boldsymbol{A} \cdot \vec{x} &= \boldsymbol{A}^T + \boldsymbol{A} \\
\nabla_{\vec{x}} (\vec{x} - \vec{y})^T \cdot \boldsymbol{A} \cdot (\vec{x} - \vec{y}) &= (\boldsymbol{A}^T + \boldsymbol{A}) \cdot (\vec{x} - \vec{y}) \\
\nabla_{\boldsymbol{A}} (\vec{x} - \vec{y})^T \cdot \boldsymbol{A} \cdot (\vec{x} - \vec{y}) &= (\vec{x} - \vec{y}) \cdot (\vec{x} - \vec{y})^T \\
\nabla_{\boldsymbol{A}} \log(\det(\boldsymbol{A})) &= \big(A^{-1}\big)^T \qquad \text{if } \boldsymbol{A} \text{ is invertible}
\end{aligned}$$

## REVERSE SWEEP

The "reverse sweep" is just an application of the chain rule. For any set of equations of the form $z = f(y_1, \ldots, y_n)$ and $y_1 = g_1(x), \ldots, y_n = g_n(x)$ it holds:

$$\frac{\partial}{\partial x} z = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \cdot \frac{\partial y_i}{\partial x}$$

*Example:*




where $\sigma(\mu) = 1/(1 + \exp(\mu))$. Thus, we obtain the following example derivatives, using the reverse sweep rule:

$$\frac{\partial z}{\partial a} = z \cdot (1 - z), \qquad \frac{\partial z}{\partial y_2} = \frac{\partial z}{\partial a} \cdot \frac{\partial a}{\partial y_2} = z \cdot (1 - z) \cdot \theta_4, \qquad \frac{\partial z}{\partial y_1} = \frac{\partial z}{\partial a} \cdot \frac{\partial a}{\partial y_1} = z \cdot (1 - z) \cdot \theta_3,$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x} = z \cdot (1 - z) \cdot \Big(\theta_3 \cdot \theta_1 + \theta_4 \cdot \theta_2\Big), \qquad \frac{\partial z}{\partial \theta_1} = \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial \theta_1} = z \cdot (1 - z) \cdot \theta_3 \cdot x$$