Script for Numerical Optimization Course B-KUL-H03E3A

Moritz Diehl
Optimization in Engineering Center (OPTEC)
and Electrical Engineering Department (ESAT-SCD), KU Leuven,
Kasteelpark Arenberg 10,
3001 Leuven, Belgium
moritz.diehl@esat.kuleuven.be

October 9, 2013

Preface

This course on numerical optimization is intended for students of mathematical engineering in the first year of their master programme, as well as for interested master and PhD students from neighboring subjects. The course's aim is to give an introduction into numerical methods for solution of optimization problems, in order to prepare the students for using and developing these methods for specific applications in engineering. The course's focus is on *continuous optimization* (rather than discrete optimization) with special emphasis on **nonlinear programming**. For this reason, the course is in large parts based on the excellent text book "Numerical Optimization" by Jorge Nocedal and Steve Wright [4]. This book appeared in Springer Verlag and is available at the ACCO book shop as well as at VTK and recommended to the students. Besides nonlinear programming, we discuss important and beautiful concepts from the field of **convex optimization** that we believe to be important to all users and developers of optimization methods. These contents and much more are covered by the equally excellent text book "Convex Optimization" by Stephen Boyd and Lieven Vandenberghe [1], that was published by Cambridge University Press (CUP). Fortunately, this book is also freely available and can be downloaded from the home page of Stephen Boyd in form of a completely legal PDF copy of the CUP book.

The course is divided into three major parts:

- Fundamental Concepts
- Unconstrained Optimization Algorithms
- Constrained Optimization Algorithms

followed by two appendices, the first containing the description of one student project done during the course exercises, and some remarks intended to help with exam preparation (including a list of questions and answers).

This script was written with major help of Jan Bouckaert (who did, among other, all the figures), David Ariens, and Laurent Sorber. Thanks go also to Dr. Carlo Savorgnan and to many students who helped with feedback and with spotting errors in the last years. Finally, the author wants to gratefully acknowledge financial support by KU Leuven's Optimization in Engineering Center OPTEC.

Moritz Diehl,
Leuven and Freiburg,
October 2013.

moritz.diehl@esat.kuleuven.be

Contents

Part I

Fundamental Concepts

Chapter 1

Fundamental Concepts of Optimization

1.1 Why Optimization?

Optimization algorithms are used in many applications from diverse areas.

- Business: Allocation of resources in logistics, investment, etc.
- Science: Estimation and fitting of models to measurement data, design of experiments.
- Engineering: Design and operation of technical systems/ e.g. bridges, cars, aircraft, digital devices, etc.

1.2 What Characterizes an Optimization Problem?

An optimization problem consists of the following three ingredients.

- An objective function, $f(x)$, that shall be minimized or maximized,
- decision variables, x, that can be chosen, and
- constraints that shall be respected, e.g. of the form $g(x) = 0$ (equality constraints) or $h(x) \geq 0$ (inequality constraints).

1.3 Mathematical Formulation in Standard Form

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) && (1.1)\\ \text{subject to} \quad & g(x) = 0, && (1.2)\\ & h(x) \geq 0. && (1.3) \end{aligned}$$

Here, $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^p$, $h : \mathbb{R}^n \to \mathbb{R}^q$, are usually assumed to be differentiable. Note that the inequalities hold for all components, i.e.

$$h(x) \geq 0 \quad \Leftrightarrow \quad h_i(x) \geq 0, \quad i = 1, \ldots, q. \tag{1.4}$$

Example 1.1 (A two dimensional example)**:**

$$\begin{aligned} \underset{x \in \mathbb{R}^2}{\text{minimize}} \quad & x_1^2 + x_2^2 && (1.5)\\ \text{subject to} \quad & x_2 - 1 - x_1^2 \geq 0, && (1.6)\\ & x_1 - 1 \geq 0. && (1.7) \end{aligned}$$

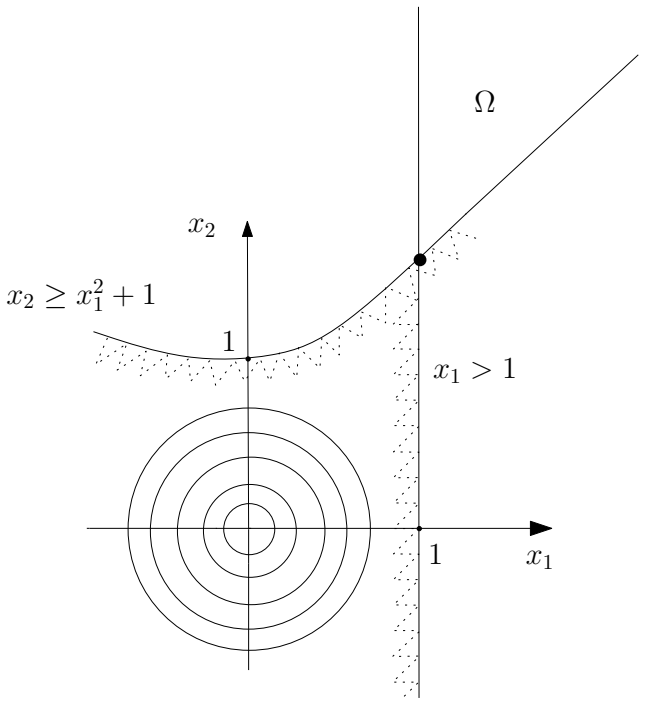


Figure 1.1: Visualization of Example 1.1, Ω is defined in Definition 1.2

1.4 Definitions

Definition 1.1
The set $\{x \in \mathbb{R}^n | f(x) = c\}$ is the "Level set" of f for the value c.

Definition 1.2
The "feasible set" Ω is $\{x \in \mathbb{R}^n | g(x) = 0,\ h(x) \geq 0\}$.

Definition 1.3
The point $x^* \in \mathbb{R}^n$ is a "global minimizer" (often also called a "global minimum") if and only if (iff) $x^* \in \Omega$ and $\forall x \in \Omega :\ f(x) \geq f(x^*)$.

Definition 1.4
The point $x^* \in \mathbb{R}^n$ is a "*strict* global minimizer" iff $x^* \in \Omega$ and $\forall x \in \Omega \setminus \{x^*\} :\ f(x) > f(x^*)$.

Definition 1.5
The point $x^* \in \mathbb{R}^n$ is a "local minimizer" iff $x^* \in \Omega$ and there exists a neighborhood $\mathcal{N}$ of x^* (e.g. an open ball around x^*) so that $\forall x \in \Omega \cap \mathcal{N} :\ f(x) \geq f(x^*)$.

Definition 1.6
The point $x^* \in \mathbb{R}^n$ is a "*strict* local minimizer" iff $x^* \in \Omega$ and there exists a neighborhood $\mathcal{N}$ of x^* so that $\forall x \in (\Omega \cap \mathcal{N}) \setminus \{x^*\} :\ f(x) > f(x^*)$.

Example 1.2 (A one dimensional example)**:** Note that this example is not convex.

$$\begin{aligned} \underset{x \in \mathbb{R}}{\text{minimize}} \quad & \sin(x)\exp(x) && && (1.8)\\ \text{subject to} \quad & x && \geq\ 0, && (1.9)\\ & x && \leq\ 4\pi. && (1.10) \end{aligned}$$

- $\Omega = \{x \in \mathbb{R} | x \geq 0,\ x \leq 4\pi\} = [0, 4\pi]$
- Three local minimizers (which?)
- One global minimizer (which?)

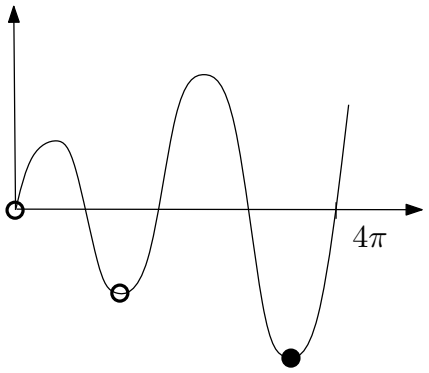


Figure 1.2: Visualization of Example 1.2

1.5 When Do Minimizers Exist?

Theorem 1.1 (Weierstrass)**:** *If $\Omega \subset \mathbb{R}^n$ is compact (i.e. bounded and closed) and $f : \Omega \to \mathbb{R}$ is continuous then there exists a global minimizer of the optimization problem*

$$\underset{x \in \mathbb{R}^n}{\textit{minimize}} \quad f(x) \quad \textit{subject to} \quad x \in \Omega. \tag{1.11}$$

Proof. Regard the graph of f, $G = \{(x, s) \in \mathbb{R}^n \times \mathbb{R} | x \in \Omega,\ s = f(x)\}$. G is a compact set, and so is the projection of G onto its last coordinate, the set $\tilde{G} = \{s \in \mathbb{R} | \exists x \text{ such that } (x, s) \in G\}$, which is a compact interval $[f_{\min}, f_{\max}] \subset \mathbb{R}$. By construction, there must be at least one x^* so that $(x^*, f_{\min}) \in G$. □

Thus, minimizers exist under fairly mild circumstances. Though the proof was constructive, it does not lend itself to an efficient algorithm. The topic of this lecture is how to practically find minimizers with help of computer algorithms.

Chapter 2

Types of Optimization Problems

In order to choose the right algorithm for a practical problem, we should know how to classify it and which mathematical structures can be exploited. Replacing an inadequate algorithm by a suitable one can make solution times many orders of magnitude shorter.

2.1 Nonlinear Programming (NLP)

In this lecture we mainly treat algorithms for general Nonlinear Programming Problems or Nonlinear Programs (NLP), which are given in the form

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) && \text{(2.1a)} \\ \text{subject to} \quad & g(x) = 0, && \text{(2.1b)} \\ & h(x) \geq 0, && \text{(2.1c)} \end{aligned}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^p$, $h : \mathbb{R}^n \to \mathbb{R}^q$, are assumed to be continuously differentiable at least once, often twice and sometimes more. Differentiability of all problem functions allows us to use algorithms that are based on derivatives, in particular the so called "Newton-type optimization methods" which are the main topic of this course.

But many problems have more structure, which we should recognize and exploit in order to solve problems faster.

2.2 Linear Programming (LP)

When the functions f, g, h are affine in the general formulation (2.1), the general NLP gets something easier to solve, namely a Linear Program (LP). Explicitly, an LP can be written as follows.

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & c^T x && \text{(2.2a)} \\ \text{subject to} \quad & Ax - b = 0, && \text{(2.2b)} \\ & Cx - d \geq 0. && \text{(2.2c)} \end{aligned}$$

Here, the problem data is given by $c \in \mathbb{R}^n, A \in \mathbb{R}^{p \times n}, b \in \mathbb{R}^p, C \in \mathbb{R}^{q \times n}$, and $d \in \mathbb{R}^q$. Note that we could also have a constant contribution to the objective, i.e. have $f(x) = c^T x + c_0$, but that this would not change the minimizers x^*.

LPs can be solved very efficiently since the 1940's, when G. Dantzig invented the famous "simplex method", an "active set method", which is still widely used, but got an equally efficient competitor in the so called "interior point methods". LPs can nowadays be solved even if they have millions of variables and constraints. Every business student knows how to use them, and LPs arise in myriads of applications. LP algorithms are not treated in detail in this lecture, but please recognize them if you encounter them in practice and use the right software.

Example 2.1 (LP resulting from oil shipment cost minimization)**:** We regard a typical logistics problem that an oil production and distribution company might encounter. We want to minimize the costs of transporting oil from the oil producing countries to the oil consuming countries, as visualized in Figure 2.1.

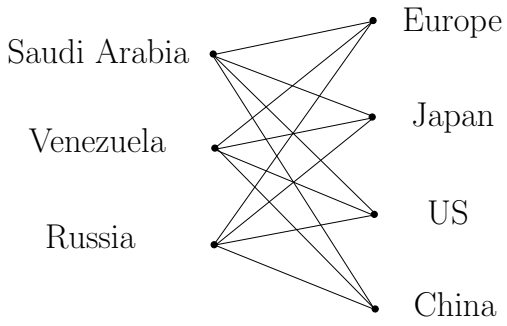


Figure 2.1: A traditional example of a LP problem: minimize the oil shipment costs while satisfying the demands on the right and not exceeding the production capabilities on the left.

More specifically, given a set of n oil production facilities with production capacities p_i with $i = 1, \ldots, n$, and given a set of m customer locations with oil demands d_j with $j = 1, \ldots, m$, and given shipment costs c_{ij} for all possible routes between each i and j, we want to decide how much oil should be transported along each route. These quantities, which we call x_{ij}, are our decision variables, in total nm real valued variables. The problem can be written as the following linear program.

$$
\begin{aligned}
\underset{x\in\mathbb{R}^{n\times m}}{\text{minimize}} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{m} x_{ij} \le p_i, \quad i = 1,\dots,n, \\
& \sum_{i=1}^{n} x_{ij} \ge d_j, \quad j = 1,\dots,m, \\
& x_{ij} \ge 0, \quad i = 1,\dots,n, \quad j = 1,\dots,m.
\end{aligned}
$$

Software for solving linear programs: CPLEX, SOPLEX, lp_solve, lingo. MATLAB: linprog.

2.3 Quadratic Programming (QP)

If in the general NLP formulation (2.1) the constraints g, h are affine (as for an LP), but the objective is a linear-quadratic function, we call the resulting problem a Quadratic Programming Problem or Quadratic Program (QP). A general QP can be formulated as follows.

$$
\begin{aligned}
\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & c^T x + \frac{1}{2}x^T B x && \text{(2.3a)} \\
\text{subject to} \quad & Ax - b = 0, && \text{(2.3b)} \\
& Cx - d \ge 0. && \text{(2.3c)}
\end{aligned}
$$

Here, in addition to the same problem data as in the LP $c \in \mathbb{R}^n, A \in \mathbb{R}^{p\times n}, b \in \mathbb{R}^p, C \in \mathbb{R}^{q\times n}, d \in \mathbb{R}^q$, we also have the "Hessian matrix" $B \in \mathbb{R}^{n\times n}$. Its name stems from the fact that $\nabla^2 f(x) = B$ for $f(x) = c^T x + \frac{1}{2}x^T B x$.

Definition 2.1 (Convex QP)
If the Hessian matrix B is positive semi-definite (i.e. if $\forall z \in \mathbb{R}^n : \; z^T B z \ge 0$) we call the QP (2.3) a "convex QP". Convex QPs are tremendously easier to solve globally than "non-convex QPs" (i.e., where the Hessian B is not positive semi-definite), which might have different local minima (i.e. have a non-convex solution set, see next section).

Definition 2.2 (Strictly convex QP)
If the Hessian matrix B is positive definite (i.e. if $\forall z \in \mathbb{R}^n \setminus \{0\} : \; z^T B z > 0$) we call the QP (2.3) a "strictly convex QP". Strictly convex QPs are a subclass of convex QPs, but often still a bit easier to solve than not-strictly convex QPs.

Example 2.2 (A non-convex QP)**:**

$$\begin{aligned} \underset{x \in \mathbb{R}^2}{\text{minimize}} \quad & \begin{bmatrix} 0 & 2 \end{bmatrix} x + \frac{1}{2} x^T \begin{bmatrix} 5 & 0 \\ 0 & -1 \end{bmatrix} x && (2.4) \\ \text{subject to} \quad & -1 \leq x_1 \leq 1, && (2.5) \\ & -1 \leq x_2 \leq 10. && (2.6) \end{aligned}$$

This problem has local minimizers at $x_a^* = (0, -1)^T$ and $x_b^* = (0, 10)^T$, but only x_b^* is a global minimizer.

Example 2.3 (A strictly convex QP)**:**

$$\begin{aligned} \underset{x \in \mathbb{R}^2}{\text{minimize}} \quad & \begin{bmatrix} 0 & 2 \end{bmatrix} x + \frac{1}{2} x^T \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} x && (2.7) \\ \text{subject to} \quad & -1 \leq x_1 \leq 1, && (2.8) \\ & -1 \leq x_2 \leq 10. && (2.9) \end{aligned}$$

This problem has only one (strict) local minimizer at $x^* = (0, -1)^T$ that is also global minimizer.

Software for solving quadratic programs: CPLEX, MOSEK, qpOASES (open), OOQP (open), MATLAB: quadprog.

2.4 General Convex Optimization Problems

"The great watershed in optimization is not between linearity and nonlinearity, but convexity and nonconvexity"

R. Tyrrell Rockafellar

Both, LPs and convex QPs, are part of an important class of optimization problems, namely the "convex optimization problems". In order to define them and understand why they are so important, we first recall what is a convex set and a convex function.

Definition 2.3 (Convex Set)
A set $\Omega \subset \mathbb{R}^n$ is convex if

$$\forall x, y \in \Omega, t \in [0, 1] : \; x + t(y - x) \in \Omega. \tag{2.10}$$

("all connecting lines lie inside set")

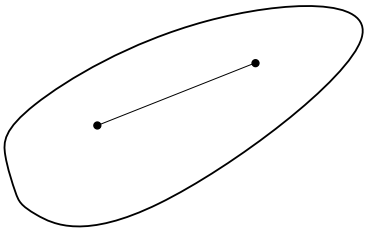


Figure 2.2: An example of a convex set

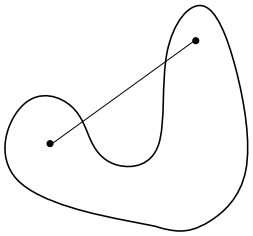


Figure 2.3: An example of a non convex set

Definition 2.4 (Convex Function)
A function $f : \Omega \to \mathbb{R}$ is convex, if Ω is convex and if

$$\forall x, y \in \Omega, t \in [0,1] : \ f(x + t(y - x)) \leq f(x) + t(f(y) - f(x)). \tag{2.11}$$

("all secants are above graph"). This definition is equivalent to saying that the Epigraph of f, i.e. the set $\{(x, s) \in \mathbb{R}^n \times \mathbb{R} | x \in \Omega, \ s \geq f(x)\}$, is a convex set.

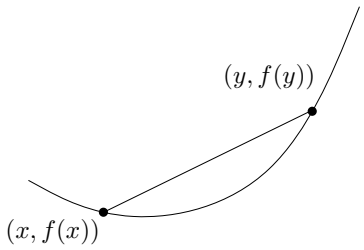


Figure 2.4: For a convex function, the line segment between any two points on the graph lies above the graph

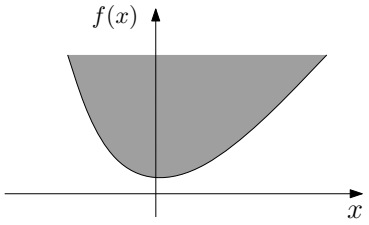


Figure 2.5: For a convex function, the Epigraph of the function (grey color) is always convex

Definition 2.5 (Convex Optimization Problem)
An optimization problem with convex feasible set Ω and convex objective function $f : \Omega \to \mathbb{R}$ is called a "convex optimization problem".

Theorem 2.1 (Local Implies Global Optimality for Convex Problems)**:** *For a convex optimization problem, every local minimum is also a global one.*

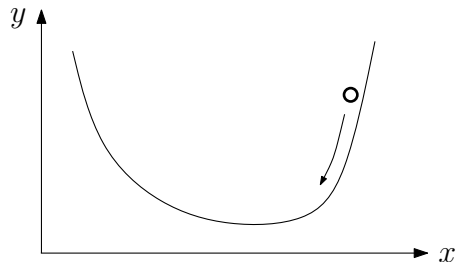


Figure 2.6: Every local minimum is also a global one for a convex function

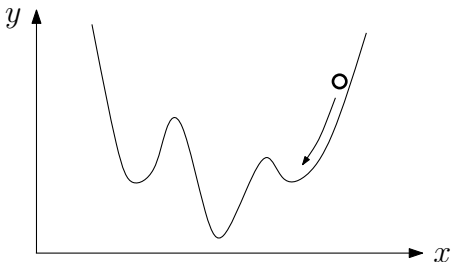


Figure 2.7: Not every local minimum is also a global one for this nonconvex function

Proof. Regard a local minimum x^* of the convex optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) \text{ s.t. } x \in \Omega.$$

We will show that for any given point $y \in \Omega$ holds $f(y) \geq f(x^*)$. Regard Figure 2.8 for a visualization of the proof.

First we choose, using local optimality, a neighborhood $\mathcal{N}$ of x^* so that for all $\tilde{x} \in \Omega \cap \mathcal{N}$ holds $f(\tilde{x}) \geq f(x^*)$. Second, we regard the connecting line between x^* and y. This line is completely contained in Ω due to convexity of Ω. Now we choose a point $\tilde{x}$ on this line that is in the neighboorhood $\mathcal{N}$, but not equal to x^*, i.e. we have $\tilde{x} = x^* + t(y - x^*)$ with $t > 0, t \leq 1$, and $\tilde{x} \in \Omega \cap \mathcal{N}$. Due to local optimality, we have $f(x^*) \leq f(\tilde{x})$, and due to convexity we have

$$f(\tilde{x}) = f(x^* + t(y - x^*)) \leq f(x^*) + t(f(y) - f(x^*)).$$

It follows that $t(f(y) - f(x^*)) \geq 0$ with $t > 0$, implying $f(y) - f(x^*) \geq 0$, as desired. □

We will discuss convexity in more detail in the following chapter.

Software for solving convex optimization problems: An environment to formulate and solve general convex optimization problems in MATLAB is CVX. On the other hand, many very specific solvers exist for more specific convex problems. For LPs and QPs we gave already software above, while many additional solvers for more general convex problem are conveniently accessible via YALMIP (which contains solvers such as SDPT3, SeDuMi and nearly all the ones mentioned above).

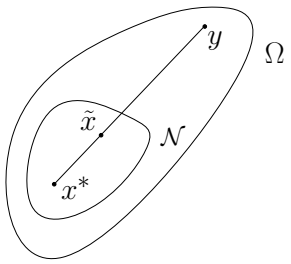


Figure 2.8: Visualization for the proof of Theorem 2.1.

2.5 Unconstrained Optimization Problems

Any NLP without constraints is called an "unconstrained optimization problem". It has the general form

$$\min_{x \in \mathbb{R}^n} f(x), \tag{2.12}$$

with usually once or twice differentiable objective function f. Unconstrained nonlinear optimization will be the focus of Part II of this lecture, while general constrained optimization problems are the focus of Part III.

2.6 Non-Differentiable Optimization Problems

If one or more of the problem functions f, g, h are not differentiable in an optimization problem (2.1), we speak of a "non-differentiable" or "non-smooth" optimization problem. Non-differentiable optimization problems are much harder to solve than general NLPs. A few solvers exist (Microsoft Excel solver, Nelder-Mead method, random search, genetic algorihms...), but are typically orders of magnitude slower than derivative-based methods (which are the topic of this course).

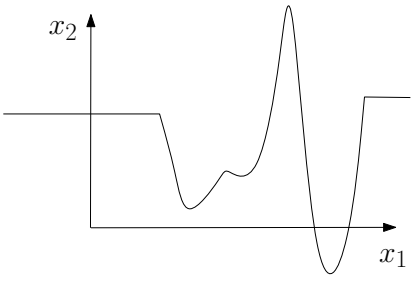


Figure 2.9: Visualization of a non-smooth objective.

2.7 Mixed-Integer Programming (MIP)

A mixed-integer programming problem or mixed-integer program (MIP) is a problem with both real and integer decision variables. A MIP can be formulated as follows:

$$\begin{aligned} \underset{\substack{x\in\mathbb{R}^n \\ z\in\mathbb{Z}^m}}{\text{minimize}} \quad & f(x,z) && && (2.13a)\\ \text{subject to} \quad & g(x,z) &=& 0, && (2.13b)\\ & h(x,z) &\geq& 0. && (2.13c) \end{aligned}$$

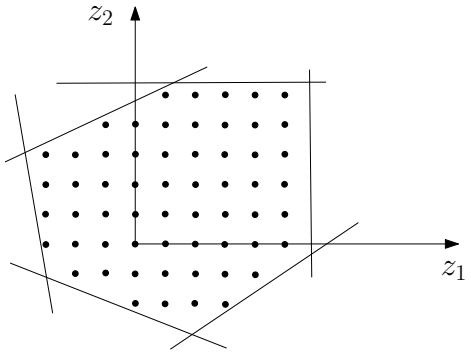


Figure 2.10: Visualization of the feasible set of an integer problem with linear constraints.

Definition 2.6 (Mixed-integer non-linear program (MINLP))
If f, g, h are twice differentiable in x and z we speak of a mixed integer non-linear program. Generally speaking, these problems are very hard to solve, due to the combinatorial nature of the variables z.

However, if a *relaxed problem*, where the variables z are no longer restricted to the integers, but to the real numbers, is convex, often very efficient solution algorithms exist. More specifically, we would require that the following problem is convex:

$$\begin{aligned} \underset{\substack{x\in\mathbb{R}^n \\ z\in\mathbb{R}^m}}{\text{minimize}} \quad & f(x,z) && && (2.14a)\\ \text{subject to} \quad & g(x,z) &=& 0, && (2.14b)\\ & h(x,z) &\geq& 0. && (2.14c) \end{aligned}$$

The efficient solution algorithms are often based on the technique of "branch-and-bound", which uses partially relaxed problems where some of the z are fixed to specific integer values and some of them are relaxed. This technique then exploits the fact that the solution of the relaxed solutions can only be better than the best integer solution. This way, the search through the combinatorial tree can be made more efficient than pure enumeration. Two important examples of such problems are given in the following.

Definition 2.7 (Mixed-integer linear program (MILP))
If f, g, h are affine in both x and z we speak of a mixed-integer linear program. These problems can efficiently be solved with codes such as the commercial code `CPLEX` or the free code `lp_solve` with a nice manual `http://lpsolve.sourceforge.net/5.5/`. A famous problem in this class is the "travelling salesman problem", which has only discrete decision variables. Linear integer programming is often just called "Integer programming (IP)". It is one of the largest research areas in the discrete optimization community. They can be solved e.g. by CPLEX or lpsolve.

Definition 2.8 (Mixed-integer quadratic programs (MIQP))
If g, h are affine and f convex quadratic in both x and z we speak of a mixed integer QP (MIQP). These problems are also efficiently solvable, mostly by commercial solvers (e.g. CPLEX).

Chapter 3

Convex Optimization

We have already discovered the favourable fact that a convex optimization problem has no local minima that are not also global. But how can we detect convexity of functions or sets?

3.1 How to Check Convexity of Functions?

Theorem 3.1 (Convexity for C^1 Functions)**:** *Assume that $f : \Omega \to \mathbb{R}$ is continuously differentiable and Ω convex. Then holds that f is convex if and only if*

$$\forall x, y \in \Omega : \quad f(y) \geq f(x) + \nabla f(x)^T (y - x). \tag{3.1}$$

i.e. tangents lie below the graph.

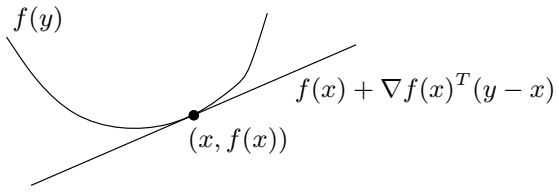


Figure 3.1: If f is convex and differentiable, then $f(y) \geq f(x) + \nabla f(x)^T (y - x)$ for all $x, y \in \Omega$

Proof. "⇒": Due to convexity of f holds for given $x, y \in \Omega$ and for any $t \in [0, 1]$ that

$$f(x + t(y - x)) - f(x) \leq t(f(y) - f(x))$$

and therefore that

$$\nabla f(x)^T (y - x) = \lim_{t \to 0} \frac{f(x + t(y - x)) - f(x)}{t} \leq f(y) - f(x).$$

"⇐": To prove that for $z = x+t(y-x) = (1-t)x+ty$ holds that $f(z) \leq (1-t)f(x)+tf(y)$ let us use Eq. (3.1) twice at z, in order to obtain $f(x) \geq f(z)+\nabla f(z)^T(x-z)$ and $f(y) \geq f(z)+\nabla f(z)^T(y-z)$ which yield, when weighted with $(1-t)$ and t and added to each other

$$(1-t)f(x)+tf(y) \geq f(z)+\nabla f(z)^T \underbrace{[(1-t)(x-z)+t(y-z)]}_{=(1-t)x+ty-z=0}.$$

□

Definition 3.1 (Generalized Inequality for Symmetric Matrices)
We write for a symmetric matrix $B = B^T$, $B \in \mathbb{R}^{n\times n}$ that "$B \succcurlyeq 0$" if and only if B is *positive semi-definite* i.e., $\forall z \in \mathbb{R}^n : \; z^T B z \geq 0$, or, equivalently, if all (real) eigenvalues of the symmetric matrix B are non-negative:

$$B \succcurlyeq 0 \iff \min \operatorname{eig}(B) \geq 0.$$

We write for two such symmetric matrices that "$A \succcurlyeq B$" iff $A - B \succcurlyeq 0$, and "$A \preccurlyeq B$" iff $B \succcurlyeq A$. We say $B \succ 0$ iff B is *positive definite*, i.e. $\forall z \in \mathbb{R}^n \setminus \{0\} : z^T B z > 0$, or that all eigenvalues of B are positive

$$B \succ 0 \iff \min \operatorname{eig}(B) > 0.$$

Definition 3.2 (Definition of $O(\cdot)$ and $o(\cdot)$)
For a function $f : \mathbb{R}^n \to \mathbb{R}^m$, we write $f(x) = O(g(x))$ iff there exists a constant $C > 0$ and a neighborhood $\mathcal{N}$ of 0 so that

$$\forall x \in \mathcal{N} : \; \|f(x)\| \leq C g(x). \tag{3.2}$$

We write $f(x) = o(g(x))$ iff there exists a neighborhood $\mathcal{N}$ of 0 and a function $c : \mathcal{N} \to \mathbb{R}$ with $\lim_{x\to 0} c(x) = 0$ so that

$$\forall x \in \mathcal{N} : \; \|f(x)\| \leq c(x) g(x). \tag{3.3}$$

In a sloppy way, we could say for $O(\cdot)$: "f shrinks as fast as g", for $o(\cdot)$: "f shrinks faster than g".

Theorem 3.2 (Convexity for C^2 Functions)**:** *Assume that $f : \Omega \to \mathbb{R}$ is twice continuously differentiable and Ω convex and open. Then holds that f is convex if and only if for all $x \in \Omega$ the Hessian is positive semi-definite, i.e.*

$$\forall x \in \Omega : \quad \nabla^2 f(x) \succcurlyeq 0. \tag{3.4}$$

Proof. To prove (3.1) ⇒ (3.4) we use a second order Taylor expansion of f at x in an arbitrary direction p:

$$f(x+tp) = f(x) + t\nabla f(x)^T p + \frac{1}{2} t^2 p^T \nabla^2 f(x) p + o(t^2 \|p\|^2).$$

From this we obtain

$$p^T\nabla^2 f(x)p = \lim_{t\to 0}\frac{2}{t^2}\underbrace{\big(f(x+tp) - f(x) - t\nabla f(x)^T p\big)}_{\geq 0,\text{ because of (3.1).}} \geq 0.$$

Conversely, to prove (3.1) $\Leftarrow$ (3.4) we use the Taylor rest term formula with some $\theta \in [0,1]$.

$$f(y) = f(x) + \nabla f(x)^T(y-x) + \underbrace{\frac{1}{2}t^2(y-x)^T\nabla^2 f(x+\theta(y-x))(y-x)}_{\geq 0,\text{ due to (3.4).}}.$$

□

Example 3.1 (Exponential Function)**:** The function $f(x) = \exp(x)$ is convex because $f''(x) = f(x) \geq 0\ \forall x \in \mathbb{R}$.

Example 3.2 (Quadratic Function)**:** The function $f(x) = c^T x + \frac{1}{2}x^T Bx$ is convex if and only if $B \succcurlyeq 0$, because $\forall x \in \mathbb{R}^n :\ \nabla^2 f(x) = B$.

Example 3.3 (The function)**:** $f(x,t) = \frac{x^T x}{t}$ is convex on $\Omega = \mathbb{R}^n \times (0,\infty)$ because its Hessian

$$\nabla^2 f(x,t) = \begin{bmatrix} \frac{2}{t}\mathbb{I}_n & -\frac{2}{t^2}x \\ -\frac{2}{t^2}x^T & \frac{2}{t^3}x^T x \end{bmatrix}$$

is positive definite. To see this, multiply it from left and right with $v = (z^T, s)^T \in \mathbb{R}^{n+1}$ which yields $v^T\nabla^2 f(x,t)v = \frac{2}{t^3}\|tz - sx\|_2^2 \geq 0$ if $t > 0$.

Definition 3.3 (Concave Function)
A function $f : \Omega \to \mathbb{R}$ is called "concave" if $-f$ is convex.

Definition 3.4 (Convex Maximization Problem)
A maximization problem

$$\max_{x\in\mathbb{R}^n} f(x) \quad \text{s.t.} \quad x \in \Omega$$

is called a "convex maximization problem" if Ω is convex and f concave. It is obviously equivalent to the convex minimization problem

$$\min_{x\in\mathbb{R}^n} -f(x) \quad \text{s.t.} \quad x \in \Omega$$

3.2 Which Sets are Convex, and which Operations Preserve Convexity?

Theorem 3.3 (Convexity of Sublevel Sets)**:** *The sublevel set* $\{x \in \Omega | f(x) \leq c\}$ *of a convex function* $f : \Omega \to \mathbb{R}$ *with respect to any constant* $c \in \mathbb{R}$ *is convex.*

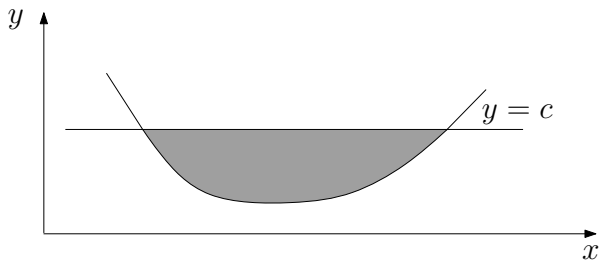


Figure 3.2: Convexity of sublevel sets

Proof. If $f(x) \leq c$ and $f(y) \leq c$ then for any $t \in [0, 1]$ holds also

$$f((1-t)x + ty) \leq (1-t)f(x) + tf(y) \leq (1-t)c + tc = c.$$

□

Several operations on convex sets preserve their convexity:

1. The intersection of finitely or infinitely many convex sets is convex.
2. Affine image: if Ω is convex, then for $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$ also the set $A\Omega + b = \{y \in \mathbb{R}^m | \exists x \in \Omega : \ y = Ax + b\}$ is convex.
3. Affine pre-image: if Ω is convex, then for $A \in \mathbb{R}^{n \times m}, b \in \mathbb{R}^n$ also the set $\{z \in \mathbb{R}^m | Az + b \in \Omega\}$ is convex.

3.3 Examples for Convex Sets

Example 3.4 (A Convex Feasible Set)**:** If $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots m$ are convex functions, then the set $\Omega = \{x \in \mathbb{R}^n | \forall i : \ f_i(x) \leq 0\}$ is a convex set, because it is the intersection of sublevel sets $\Omega = \{x \mid f_i(x) \leq 0\}$ of convex functions $f_i \Rightarrow \Omega_1, \ldots, \Omega_m$ convex $\Rightarrow \bigcap_{i=1}^{m} \Omega_i = \Omega$ convex.

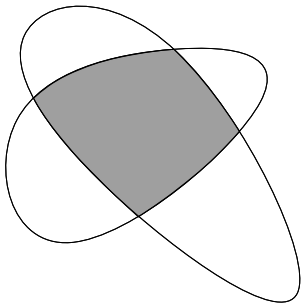


Figure 3.3: The intersection of finitely or infinitely many convex sets is convex

Example 3.5 (Linear Matrix Inequalities (LMI))**:** We define the vector space of symmetric matrices in $\mathbb{R}^{n\times n}$ as $\mathcal{S}^n = \{X \in \mathbb{R}^{n\times n} | X = X^T\}$ and the subset of positive semi-definite matrices as $\mathcal{S}^n_+ = \{X \in \mathcal{S}^n | X \succcurlyeq 0\}$. This set is convex, as can easily be checked. Let us now regard an affine map $G : \mathbb{R}^m \to \mathcal{S}^n,\ x \mapsto G(x) := A_0 + \sum_{i=1}^m A_i x_i.$, with symmetric matrices $A_0, \ldots, A_m \in \mathcal{S}^n$. The expression

$$G(x) \succcurlyeq 0$$

is called a "linear matrix inequality (LMI)". It defines a convex set $\{x \in \mathbb{R}^m | G(x) \succcurlyeq 0\}$, as the pre-image of $\mathcal{S}^n_+$ under the affine map $G(x)$.

3.4 Which Operations Preserve Convexity of Functions?

1. Affine input transformations: If $f : \Omega \to \mathbb{R}$ is convex, then also $\tilde{f}(x) = f(Ax + b)$ (with $A \in \mathbb{R}^{n\times m}$) is convex on the domain $\tilde{\Omega} = \{x \in \mathbb{R}^m | Ax + b \in \Omega\}$.

2. Concatenation with a monotone convex function: If $f : \Omega \to \mathbb{R}$ is convex and $g : \mathbb{R} \to \mathbb{R}$ is convex and monotonely increasing, then the function $g \circ f : \Omega \to \mathbb{R},\ x \mapsto g(f(x))$ is also convex.

   Proof. $\nabla^2(g \circ f)(x) = \underbrace{g''(f(x))}_{\geq 0} \underbrace{\nabla f(x) \nabla f(x)^T}_{\succcurlyeq 0} + \underbrace{g'(f(x))}_{\geq 0} \underbrace{\nabla^2 f(x)}_{\succcurlyeq 0} \succcurlyeq 0.$ □

3. The supremum over a set of convex functions $f_i(x)$, $i \in I$ is convex: $f(x) = \sup_{i\in I} f_i(x)$. This can be proven by noting that the epigraph of f is the intersection of the epigraphs of f_i.

3.5 Standard Form of a Convex Optimization Problem

In order to yield a convex feasible set Ω, the equality constraints of a convex optimization problem should only have *linear* equality constraints in order to define an affine set. Moreover, we know that a sufficient condition for a set to be convex is that it is the intersection of sublevel sets

of convex functions. This set remains convex when intersected with the affine set due to linear equality constraints. Thus, the following holds.

Theorem 3.4 (Sufficient Condition for Convex NLP)**:** *If in the NLP formulation* (2.1) *the objective f is convex, the equalities g are affine, and the inequalities h_i are concave functions, then the NLP is a convex optimization problem.*

In convex optimization texts, often a different notation for a general convex optimization problem is chosen, where the equalities are directly replaced by an affine function and the inequalities are chosen to be $\leq$ in order to be able to say that the defining functions are "convex", not "concave", just for convenience. The convex optimization problem standard form could therefore look as follows.

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f_0(x) && \text{(3.5a)} \\ \text{subject to} \quad & Ax = b && \text{(3.5b)} \\ & f_i(x) \leq 0, \quad i = 1, \ldots, m. && \text{(3.5c)} \end{aligned}$$

Here, the the above theorem can shortly be summarized as "Problem (3.5) is convex if $f_0, \ldots, f_m$ are convex.".

Example 3.6 (Quadratically Constrained Quadratic Program (QCQP))**:** A convex optimization problem of the form (3.5) with $f_i(x) = d_i + c_i^T x + \frac{1}{2} x^T B_i x$ with $B_i \succcurlyeq 0$ for $i = 0, 1, \ldots, m$ is called a "Quadratically Constrained Quadratic Program (QCQP)".

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & c_0^T x + \frac{1}{2} x^T B_0 x && \text{(3.6a)} \\ \text{subject to} \quad & Ax = b && \text{(3.6b)} \\ & d_i + c_i^T x + \frac{1}{2} x^T B_i x \leq 0, \quad i = 1, \ldots, m. && \text{(3.6c)} \end{aligned}$$

By choosing $B_1 = \ldots = B_m = 0$ we would obtain a usual QP, and by also setting $B_0 = 0$ we would obtain an LP. Therefore, the class of QCQPs contains both LPs and QPs as subclasses.

3.6 Semidefinite Programming (SDP)

An interesting class of convex optimization problems makes use of linear matrix inequalities (LMI) in order to describe the feasible set. As defined before, an LMI is a generalized form of inequality of the form

$$B_0 + \sum_{i=1}^{n} B_i x_i \succcurlyeq 0,$$

where the matrices $B_0, \ldots, B_m$ are all in the vector space $\mathcal{S}^k$ of symmetric matrices of a given dimension $\mathbb{R}^{k \times k}$.

As it involves the constraint that some matrices should remain positive semidefinite, this problem class is called "Semidefinite Programming (SDP)". A general SDP can be formulated as

$$\min_{x \in \mathbb{R}^n} \quad c^T x \tag{3.7a}$$

$$\text{subject to} \quad Ax - b = 0, \tag{3.7b}$$

$$B_0 + \sum_{i=1}^{n} B_i x_i \succcurlyeq 0. \tag{3.7c}$$

It turns out that all LPs, QPs, and QCQPs can also be formulated as SDPs, besides several other convex problems. Semidefinite Programming is a very powerful tool in convex optimization.

Example 3.7 (Minimizing Largest Eigenvalue)**:** We regard a symmetric matrix $G(x)$ that affinely depends on some design variables $x \in \mathbb{R}^n$, i.e. $G(x) = B_0 + \sum_{i=1}^{n} B_i x_i$ with $B_i \in \mathcal{S}^k$ for $i = 1, \ldots, n$. If we want to minimize the largest eigenvalue of $G(x)$, i.e. to solve

$$\min_x \lambda_{\max}\left(G(x)\right)$$

we can formulate this problem as an SDP by adding a slack variable $s \in \mathbb{R}$, as follows:

$$\min_{s \in \mathbb{R}, x \in \mathbb{R}^n} \quad s \tag{3.8a}$$

$$\text{subject to} \quad \mathbb{I}_k s - \sum_{i=1}^{n} B_i x_i - B_0 \succcurlyeq 0. \tag{3.8b}$$

Software: An excellent tool to formulate and solve convex optimization problems in a MATLAB environment is CVX, which is available as open-source code and easy to install.

3.7 An Optimality Condition for Convex Problems

Theorem 3.5 (First Order Optimality Condition for Convex Problems)**:** *Regard the convex optimization problem*

$$\min_{x \in \mathbb{R}^n} f(x) \quad s.t. \quad x \in \Omega$$

with continuously differentiable objective function f. A point $x^* \in \Omega$ is a global optimizer if and only if

$$\forall y \in \Omega: \quad \nabla f(x^*)^T (y - x^*) \geq 0. \tag{3.9}$$

Proof. If the condition holds, then due to the C^1 characterization of convexity of f in Eq. (3.1) we have for any feasible $y \in \Omega$

$$f(y) \geq f(x^*) + \underbrace{\nabla f(x^*)^T (y - x^*)}_{\geq 0} \geq f(x^*).$$

Conversely, if we assume for contradiction that we have a $y \in \Omega$ with $\nabla f(x^*)^T (y - x^*) < 0$ then we could regard a Taylor expansion

$$f(x^* + t(y - x^*)) = f(x^*) + t \underbrace{\nabla f(x^*)^T (y - x^*)}_{<0} + \underbrace{o(t)}_{\to 0}$$

yielding $f(x^* + t(y - x^*)) < f(x^*)$ for $t > 0$ small enough to let the last term be dominated by the second last one. Thus, x^* would not be a global minimizer. □

Corollary (Unconstrained Convex Problems)**:** *Regard the unconstrained problem*

$$\min_{x \in \mathbb{R}^n} f(x)$$

with $f(x)$ convex. Then a necessary and sufficient condition for x^* to be a global optimizer is

$$\nabla f(x^*) = 0. \tag{3.10}$$

Example 3.8 (Unconstrained Quadratic)**:** Regard the unconstrained problem

$$\min_{x \in \mathbb{R}^n} c^T x + \frac{1}{2} x^T B x \tag{3.11}$$

with $B \succ 0$. Due to the condition $0 = \nabla f(x^*) = c + Bx$, its unique optimizer is $x^* = -B^{-1}c$. The optimal value of (3.11) is given by the following basic relation, that we will often use in the following chapters.

$$\left(\min_{x \in \mathbb{R}^n} c^T x + \frac{1}{2} x^T B x \right) = -\frac{1}{2} c^T B^{-1} c. \tag{3.12}$$

Chapter 4

The Lagrangian Function and Duality

Let us in this section regard a (not-necessarily convex) NLP in standard form (2.1) with functions $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^p$, and $h : \mathbb{R}^n \to \mathbb{R}^q$.

Definition 4.1 (Primal Optimization Problem)
We will denote the globally optimal value of the objective function subject to the constraints as the "primal optimal value" p^*, i.e.,

$$p^* = \left(\min_{x \in \mathbb{R}^n} \; f(x) \text{ s.t. } g(x) = 0, \; h(x) \geq 0 \right), \tag{4.1}$$

and we will denote this optimization problem as the "primal optimization problem".

Definition 4.2 (Lagrangian Function and Lagrange Multipliers)
We define the so called "Lagrangian function" to be

$$\mathcal{L}(x, \lambda, \mu) = f(x) - \lambda^T g(x) - \mu^T h(x). \tag{4.2}$$

Here, we have introduced the so called "Lagrange multipliers" or "dual variables" $\lambda \in \mathbb{R}^p$ and $\mu \in \mathbb{R}^q$. The Lagrangian function plays a crucial role in both convex and general nonlinear optimization. We typically require the inequality multipliers μ to be positive, $\mu \geq 0$, while the sign of the equality multipliers λ is arbitrary. This is motivated by the following basic lemma.

Lemma 4.1 (Lower Bound Property of Lagrangian)**:** *If $\tilde{x}$ is a feasible point of* (4.1) *and $\mu \geq 0$, then*

$$\mathcal{L}(\tilde{x}, \lambda, \mu) \leq f(\tilde{x}). \tag{4.3}$$

Proof. $\mathcal{L}(\tilde{x}, \lambda, \mu) = f(\tilde{x}) - \lambda^T \underbrace{g(\tilde{x})}_{=0} - \underbrace{\mu^T}_{\geq 0} \underbrace{h(\tilde{x})}_{\geq 0} \leq f(\tilde{x})$. □

4.1 Lagrange Dual Function and Weak Duality

Definition 4.3 (Lagrange Dual Function)
We define the so called "Lagrange dual function" as the unconstrained infimum of the Lagrangian over x, for fixed multipliers λ, μ.

$$q(\lambda, \mu) = \inf_{x \in \mathbb{R}^n} \mathcal{L}(x, \lambda, \mu). \tag{4.4}$$

This function will often take the value $-\infty$, in which case we will say that the pair (λ, μ) is "dual infeasible" for reasons that we motivate in the last example of this subsection.

Lemma 4.2 (Lower Bound Property of Lagrange Dual)**:** *If $\mu \geq 0$, then*

$$q(\lambda, \mu) \leq p^* \tag{4.5}$$

Proof. The lemma is an immediate consequence of Eq. (4.3) which implies that for any feasible $\tilde{x}$ holds $q(\lambda, \mu) \leq f(\tilde{x})$. This inequality holds in particular for the global minimizer x^* (which must be feasible), yielding $q(\lambda, \mu) \leq f(x^*) = p^*$. □

Theorem 4.3 (Concavity of Lagrange Dual)**:** *The function $q : \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}$ is concave, even if the original NLP was not convex.*

Proof. We will show that $-q$ is convex. The Lagrangian $\mathcal{L}$ is an affine function in the multipliers λ and μ, which in particular implies that $-\mathcal{L}$ is convex in (λ, μ). Thus, the function $-q(\lambda, \mu) = \sup_x -\mathcal{L}(x, \lambda, \mu)$ is the supremum of convex functions in (λ, μ) that are indexed by x, and therefore convex. □

A natural question to ask is what is the best lower bound that we can get from the Lagrange dual function. We obtain it by maximizing the Lagrange dual over all possible multiplier values, yielding the so called "dual problem".

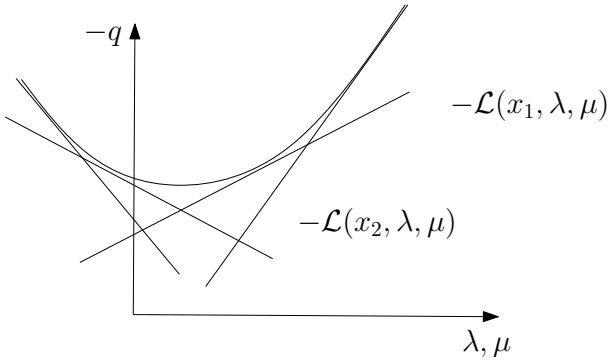


Figure 4.1: The negative dual function -q is supremum of linear functions in λ and μ hence convex.

Definition 4.4 (Dual Problem)
The "dual problem" with "dual optimal value" d^* is defined as the convex maximization problem

$$d^* = \left(\max_{\lambda \in \mathbb{R}^p, \mu \in \mathbb{R}^q} q(\lambda, \mu) \text{ s.t. } \mu \geq 0 \right) \tag{4.6}$$

It is interesting to note that the dual problem is always convex, even if the so called "primal problem" is not. As an immediate consequence of the last lemma, we obtain a very fundamental result that is called "weak duality".

Theorem 4.4 (Weak Duality)**:**

$$d^* \leq p^* \tag{4.7}$$

This theorem holds for any arbitrary optimization problem, but does only unfold its full strength in convex optimization, where very often holds a strong version of duality. We will just cite the important result here, without proof.

4.2 Strong Duality for Convex Problems

Theorem 4.5 (Strong Duality)**:** *If the primal optimization problem* (4.1) *is convex and the so called "Slater condition" holds, then primal and dual objective are equal to each other,*

$$d^* = p^*. \tag{4.8}$$

For completeness, we briefly state the technical condition used in the above theorem, which is satisfied for most convex optimization problems of interest. The proof of the theorem can for

example be found in [1].

Definition 4.5 (Slater's Constraint Qualification)
The Slater condition is satisfied for a convex optimization problem (4.1) if there exists at least one feasible point $\bar{x} \in \Omega$ such that all nonlinear inequalities are strictly satisfied. More explicitly, for a convex problem we must have affine equality constraints, $g(x) = Ax + b$, and the inequality constraint functions $h_i(x)$, $i = 1, \ldots, q$, can be either affine or concave functions, thus we can without loss of generality assume that the first $q_1 \leq q$ inequalities are affine and the remaining ones concave. Then the Slater condition holds if and only if there exists an $\bar{x}$ such that

$$\begin{aligned} A\bar{x} + b &= 0, && && (4.9a)\\ h_i(\bar{x}) &\geq 0, && \text{for} \quad i = 1, \ldots, q_1, && (4.9b)\\ h_i(\bar{x}) &> 0, && \text{for} \quad i = q_1 + 1, \ldots, q. && (4.9c) \end{aligned}$$

Note that the Slater condition is satisfied for all feasible LP and QP problems.

Strong duality allows us to reformulate a convex optimization problem into its dual, which looks very differently, but gives the same solution. We will look at this at hand of two examples.

Example 4.1 (Dual of a strictly convex QP)**:** We regard the following strictly convex QP (i.e., with $B \succ 0$)

$$\begin{aligned} p^* = \min_{x \in \mathbb{R}^n} \quad & c^T x + \frac{1}{2} x^T B x && && (4.10a)\\ \text{subject to} \quad & Ax - b &= 0, && (4.10b)\\ & Cx - d &\geq 0. && (4.10c) \end{aligned}$$

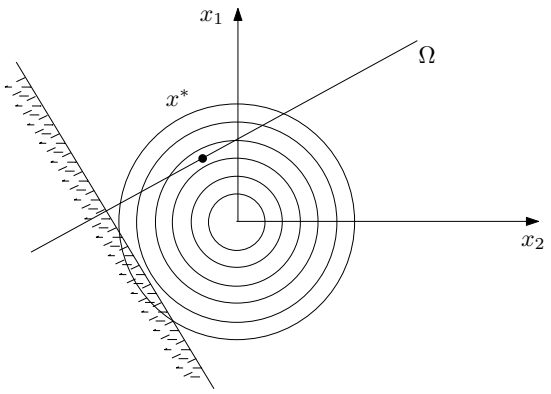


Figure 4.2: Illustration of a QP in two variables with one equality and one inequality constraint

Its Lagrangian function is given by

$$\begin{aligned}\mathcal{L}(x,\lambda,\mu) &= c^T x + \frac{1}{2}x^T B x - \lambda^T(Ax-b) - \mu^T(Cx-d) \\ &= \lambda^T b + \mu^T d + \frac{1}{2}x^T B x + \left(c - A^T\lambda - C^T\mu\right)^T x.\end{aligned}$$

The Lagrange dual function is the infimum value of the Lagrangian with respect to x, which only enters the last two terms in the above expression. We obtain

$$\begin{aligned}q(\lambda,\mu) &= \lambda^T b + \mu^T d + \inf_{x\in\mathbb{R}^n}\left(\frac{1}{2}x^T B x + \left(c - A^T\lambda - C^T\mu\right)^T x\right) \\ &= \lambda^T b + \mu^T d - \frac{1}{2}\left(c - A^T\lambda - C^T\mu\right)^T B^{-1}\left(c - A^T\lambda - C^T\mu\right)\end{aligned}$$

where we have made use of the basic result (3.12) in the last row.

Therefore, the dual optimization problem of the QP (4.10) is given by

$$\begin{aligned}d^* = \max_{\lambda\in\mathbb{R}^p,\mu\in\mathbb{R}^q} -\frac{1}{2}c^T B^{-1} c &+ \begin{bmatrix} b + AB^{-1}c \\ d + CB^{-1}c\end{bmatrix}^T \begin{bmatrix}\lambda \\ \mu\end{bmatrix} \\ &- \frac{1}{2}\begin{bmatrix}\lambda \\ \mu\end{bmatrix}^T \begin{bmatrix}A \\ C\end{bmatrix} B^{-1} \begin{bmatrix}A \\ C\end{bmatrix}^T \begin{bmatrix}\lambda \\ \mu\end{bmatrix} \qquad (4.11a)\end{aligned}$$

$$\text{subject to} \quad \mu \geq 0. \qquad (4.11b)$$

Due to the fact that the objective is concave, this problem is again a convex QP, but not a strictly convex one. Note that the first term is a constant, but that we have to keep it in order to make sure that $d^* = p^*$, i.e. strong duality, holds.

Example 4.2 (Dual of an LP)**:** Let us now regard the following LP

$$\begin{aligned}p^* = \min_{x\in\mathbb{R}^n} \quad & c^T x && (4.12a)\\ \text{subject to} \quad & Ax - b = 0, && (4.12b)\\ & Cx - d \geq 0. && (4.12c)\end{aligned}$$

Its Lagrangian function is given by

$$\begin{aligned}\mathcal{L}(x,\lambda,\mu) &= c^T x - \lambda^T(Ax-b) - \mu^T(Cx-d) \\ &= \lambda^T b + \mu^T d + \left(c - A^T\lambda - C^T\mu\right)^T x.\end{aligned}$$

Here, the Lagrange dual is

$$\begin{aligned}q(\lambda,\mu) &= \lambda^T b + \mu^T d + \inf_{x\in\mathbb{R}^n}\left(c - A^T\lambda - C^T\mu\right)^T x \\ &= \lambda^T b + \mu^T d + \begin{cases} 0 & \text{if } c - A^T\lambda - C^T\mu = 0 \\ -\infty & \text{else.}\end{cases}\end{aligned}$$

Thus, the objective function $q(\lambda, \mu)$ of the dual optimization problem is $-\infty$ at all points that do not satisfy the linear equality $c - A^T\lambda - C^T\mu = 0$. As we want to maximize, these points can be regarded as infeasible points of the dual problem (that is why we call them "dual infeasible"), and we can explicitly write the dual of the above LP (4.12) as

$$\begin{aligned} d^* = \quad & \max_{\lambda \in \mathbb{R}^p, \mu \in \mathbb{R}^q} \quad \begin{bmatrix} b \\ d \end{bmatrix}^T \begin{bmatrix} \lambda \\ \mu \end{bmatrix} && \text{(4.13a)} \\ & \text{subject to} \quad c - A^T\lambda - C^T\mu = 0, && \text{(4.13b)} \\ & \qquad\qquad\qquad\quad \mu \geq 0. && \text{(4.13c)} \end{aligned}$$

This is again an LP and it can be proven that strong duality holds for all LPs for which at least one feasible point exists, i.e. we have $d^* = p^*$, even though the two problems look quite different.

Example 4.3 (Dual Decomposition)**:** Let us regard N people that share one common resource, such as water, air, oil or electricity. The total amount of this resource is limited by M and the demand that each person requests is given by $x_i \in \mathbb{R}$ for $i = 1, \ldots, N$. The cost benefit that each person has from using the resource is given by the cost function $f_i(x_i, y_i)$, where $y_i \in \mathbb{R}^{n_i}$ are other decision variables that each person has. Let us now assume that the resource has no fixed price and instead shall be distributed in order to minimize the sum of the individual cost functions, i.e. the distributor shall solve the following optimization problem.

$$\begin{aligned} & \underset{x,y}{\text{minimize}} \quad \sum_{i=1}^{N} f_i(x_i, y_i) \\ & \text{subject to} \quad M - \sum_{i=1}^{N} x_i \geq 0 \end{aligned} \tag{4.14}$$

It is clear that depending on the size of N and the complexity of the individual cost functions f_i this can be a difficult problem to solve. Moreover, it is difficult for a central distributor to know the individual cost functions. It turns out that duality theory delivers us an elegant way to facilitate a decentralized solution to the optimization problem that is very widely used in real life. Let us introduce a Lagrange multiplier $\mu \in \mathbb{R}$ for the constraint, and form the Lagrangian function

$$\mathcal{L}(x, y, \mu) = -M\mu + \sum_{i=1}^{N} f_i(x_i, y_i) + \mu x_i.$$

The Lagrange dual function $q : \mathbb{R} \to \mathbb{R}$ is now given by

$$q(\mu) = \inf_{x,y} \mathcal{L}(x, y, \mu) = -M\mu + \sum_{i=1}^{N} \inf_{x_i, y_i} f_i(x_i, y_i) + \mu x_i.$$

It is a remarkable fact that the minimization of the global Lagrangian can be decomposed into N individual local optimizations. If in addition the functions f_i are convex, we know that maximizing the dual function is equivalent to the original problem. We only need to find the right multiplier μ^* by solving the dual problem

$$\max_{\mu \in \mathbb{R}} q(\mu) \quad \text{subject to} \quad \mu \geq 0.$$

The fact that q can be evaluated by N parallel minimizations can be beneficial for distribution of the computational load. This overall method to solve a large scale optimization problem is also known as *dual decomposition*. The local optimization problems are solved by so called *local agents*, while the Lagrange multiplier μ is determined by the so called *central agent*. It is instructive to see what is the local optimization problem that each local agent has to solve at the optimal multiplier μ^* (let us assume the minimum is attained so that we can replace the infimum by the minimum):

$$\min_{x_i, y_i} f_i(x_i, y_i) + \mu^* x_i$$

Here, the original cost function f_i is augmented by the amount of resource, x_i, multiplied with a non-negative factor μ^*. The more of the resource the actor consumes, the more its local cost function is penalized by $\mu^* x_i$. This is exactly what would happen if each actor would have to pay a price μ^* for each unit of resource. We see that the role of Lagrange multipliers and of prices is very similar. For this reason, the multipliers are sometimes also called *shadow prices*, and dual decomposition is sometimes called *price decomposition*. One way of making sure that the global optimization problem is solved is to find out what is the right price and then ask each actor to pay for using the resource. Finding the right price is a difficult problem, of course, and not solving it exactly, or the slow convergence of μ towards μ^* or its possible oscillations during this process are one of the prime reasons for macroeconomic difficulties.

To see that setting the right price μ^* really solves the problem of sharing the resource in an optimal way, let us in addition assume that the local cost functions f_i are strictly convex. This implies that q is differentiable. As it is concave as well, $-q$ is convex, and we can apply the optimality condition of convex problems from the last chapter: at the maximum μ^* must hold

$$-\nabla_\mu q(\mu^*)^T(\mu - \mu^*) \geq 0 \quad \forall \mu \geq 0. \tag{4.15}$$

If we assume that the optimal price μ^* is nonzero, this implies that $\nabla q(\mu^*) = 0$, or differently written, $\frac{\partial q}{\partial \mu}(\mu^*) = 0$. Let us compute the derivative of q w.r.t. μ explicitly. First, also assume that f_i are differentiable, and that the optimal local solutions $x_i^*(\mu)$ and $y_i^*(\mu)$ depend differentiably on μ. We then have that q is given by

$$q(\mu) = -M\mu + \sum_{i=1}^{N} f_i(x_i^*(\mu), y_i^*(\mu)) + \mu x_i^*(\mu).$$

Its derivative can be computed easily by using the following observation that follows from optimality of $x_i^*(\mu)$ and $y_i^*(\mu)$:

$$\frac{d}{d\mu} f_i(x_i^*(\mu), y_i^*(\mu)) = \underbrace{\frac{\partial f_i}{\partial x_i}(x_i^*(\mu), y_i^*(\mu))}_{=0} \frac{\partial}{\partial \mu} x_i^*(\mu) + \underbrace{\frac{\partial f_i}{\partial y_i}(x_i^*(\mu), y_i^*(\mu))}_{=0} \frac{\partial}{\partial \mu} y_i^*(\mu) = 0.$$

Thus, the derivative of q is simply given by

$$\frac{\partial q}{\partial \mu}(\mu) = -M + \sum_{i=1}^{N} x_i^*(\mu).$$

Dual optimality in the considered case of a positive price $\mu^* > 0$ is thus equivalent to

$$\sum_{i=1}^{N} x_i^*(\mu^*) = M.$$

We see that, if the positive price μ^* is set in the optimal way, the resource is exactly used up to its limit M. On the other hand, if the optimal solution would for some strange reason be given by $\mu^* = 0$, the optimality condition in Equation (4.15) would imply that $\frac{\partial q}{\partial \mu}(0) \leq 0$, or equivalently,

$$\sum_{i=1}^{N} x_i^*(0) \leq M.$$

For an optimal allocation, only resources that are not used up to their limits should have a zero price. Conversely, if the local optimizations with zero price yield a total consumption larger than the limit M, this is a clear indication that the resource should have a positive price instead.

An algorithm that can be interpreted as a variant of the gradient algorithm in the next chapter is the following: when the gradient $\frac{\partial q}{\partial \mu}(\mu) = -M + \sum_{i=1}^{N} x_i^*(\mu)$ is positive, i.e. when the resource is used more than M, we increase the price μ, and when the gradient is negative, i.e. not the full capacity M of the resource is used, we decrease the price.

Part II

Unconstrained Optimization Algorithms

Chapter 5

Optimality Conditions

In this part of the course we regard unconstrained optimization problems of the form

$$\min_{x \in D} f(x), \tag{5.1}$$

where we regard objective functions $f : D \to \mathbb{R}$ that are defined on some open domain $D \subset \mathbb{R}^n$. We are only interested in minimizers that lie inside of D. We might have $D = \mathbb{R}^n$, but often this is not the case, e.g. as in the following example:

$$\min_{x \in (0,\infty)} \frac{1}{x} + x. \tag{5.2}$$

5.1 Necessary Optimality Conditions

Theorem 5.1 (First Order Necessary Conditions (FONC))**:** *If $x^* \in D$ is local minimizer of $f : D \to \mathbb{R}$ and $f \in C^1$ then*

$$\nabla f(x^*) = 0. \tag{5.3}$$

Proof. Let us assume for contradiction that $\nabla f(x^*) \neq 0$. Then $p = -\nabla f(x^*)$ would be a descent direction in which the objective could be improved, as follows: As D is open and $f \in C^1$, we could find a $t > 0$ that is small enough so that for all $\tau \in [0, t]$ holds $x^* + \tau p \in D$ and $\nabla f(x^* + \tau p)^T p < 0$. By Taylor's Theorem, we would have for some $\theta \in (0, t)$ that

$$f(x^* + tp) = f(x^*) + t \underbrace{\nabla f(x^* + \theta p)^T p}_{<0} < f(x^*).$$

□

Definition 5.1 (Stationary Point)
A point $\bar{x}$ with $\nabla f(\bar{x}) = 0$ is called a *stationary point* of f.

Definition 5.2 (Descent Direction)
A vector $p \in \mathbb{R}^n$ with $\nabla f(x)^T p < 0$ is called a *descent direction* at x.

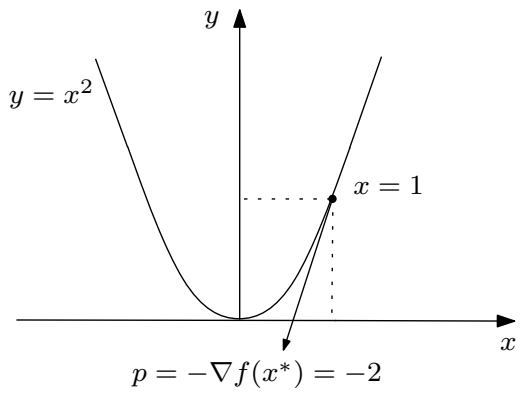


Figure 5.1: An illustration of a descent direction for $f(x) = x^2$

Theorem 5.2 (Second Order Necessary Conditions (SONC))**:** *If $x^* \in D$ is local minimizer of $f : D \to \mathbb{R}$ and $f \in C^2$ then*

$$\nabla^2 f(x^*) \succcurlyeq 0. \tag{5.4}$$

Proof. If (5.4) would not hold there would be a $p \in \mathbb{R}^n$ so that $p^T \nabla^2 f(x^*) p < 0$. Then the objective could be improved in direction p, by choosing again a sufficiently small $t > 0$ so that for all $\tau \in [0, t]$ holds $p^T \nabla^2 f(x^* + \tau p) p < 0$. By Taylor's Theorem, we would have for some $\theta \in (0, t)$ that

$$f(x^* + tp) = f(x^*) + \underbrace{t \nabla f(x^*)^T p}_{=0} + \frac{1}{2} t^2 \underbrace{p^T \nabla^2 f(x^* + \theta p) p}_{<0} < f(x^*).$$

□

Note that the second order necessary condition (5.4) is not sufficient for a stationary point x^* to be a minimizer. This is illustrated by the function $f(x) = x^3$ or $f(x) = -x^4$ which are saddle points and maximizers respectively, both fulfilling SONC.

5.2 Sufficient Optimality Conditions

For convex functions, we have already proven the following result.

Theorem 5.3 (Convex First Order Sufficient Conditions (cFOSC))**:** *Assume that $f : D \to \mathbb{R}$ is C^1 and convex. If $x^* \in D$ is a stationary point of f, then x^* is a global minimizer of f.*

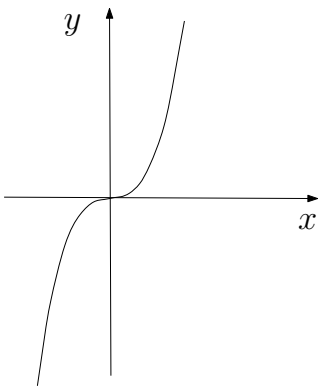


Figure 5.2: Stationary points are not always optimal

How can we obtain a sufficient optimality condition for general nonlinear, but smooth functions f?

Theorem 5.4 (Second Order Sufficient Conditions (SOSC))**:** *Assume that $f : D \to \mathbb{R}$ is C^2. If $x^* \in D$ is a stationary point and*

$$\nabla^2 f(x^*) \succ 0. \tag{5.5}$$

then x^* is a strict local minimizer of f.

Proof. We can choose a sufficiently small closed ball B around x^* so that for all $x \in B$ holds $\nabla^2 f(x) \succ 0$. Restricted to this ball, we have a convex problem, so that the previous Theorem 5.3 together with stationarity of x^* yields that x^* is a minimizer within this ball, i.e. a local minimizer. To prove that it is strict, we look for any $x \in B \backslash x*$ at the Taylor expansion, which yields with some $\theta \in (0, 1)$

$$f(x) = f(x^*) + \underbrace{\nabla f(x^*)^T (x - x^*)}_{=0} + \frac{1}{2} \underbrace{(x - x^*)^T \nabla^2 f(x^* + \theta(x - x^*))(x - x^*)}_{>0} > f(x^*).$$

□

Note that the second order sufficient condition (5.5) is not necessary for a stationary point x^* to be a strict local minimizer. This is illustrated by the function $f(x) = x^4$ for which $x^* = 0$ is a strict local minimizer with $\nabla^2 f(x^*) = 0$.

5.3 Perturbation Analysis

In numerical mathematics, we can never evaluate functions at precisions higher than machine precision. Thus, we usually compute only solutions to slightly perturbed problems, and are most

interested in minimizers that are stable against small perturbations. This is the case for strict local minimizers that satisfy the second order sufficient condition (5.5).

For this aim we regard functions $f(x, a)$ that depend not only on $x \in \mathbb{R}^n$ but also on some "disturbance parameter" $a \in \mathbb{R}^m$. We are interested in the parametric family of problems $\min_x f(x, a)$ yielding minimizers $x^*(a)$ depending on a.

Definition 5.3 (Solution map)
For a parametric optimisation problem

$$\min_{x \in D} f(x, a) \tag{5.6}$$

the dependency of x^* on a in the neighborhood of a fixed value $\bar{a}$, $x^*(a)$ is called the solution map.

Theorem 5.5 (Stability of Parametric Solutions)**:** *Assume that $f : D \times \mathbb{R}^m \to \mathbb{R}$ is C^2, and regard the minimization of $f(\cdot, \bar{a})$ for a given fixed value of $\bar{a} \in \mathbb{R}^m$. If $\bar{x} \in D$ satisfies the SOSC condition, i.e. $\nabla_x f(\bar{x}, \bar{a}) = 0$ and $\nabla_x^2 f(\bar{x}, \bar{a}) \succ 0$, then there is a neighborhood $\mathcal{N} \subset \mathbb{R}^m$ around $\bar{a}$ so that the parametric minimizer function $x^*(a)$ is well defined for all $a \in \mathcal{N}$, is differentiable in $\mathcal{N}$, and $x^*(\bar{a}) = \bar{x}$. Its derivative at $\bar{a}$ is given by*

$$\frac{\partial(x^*(\bar{a}))}{\partial a} = -\Big(\nabla_x^2 f(\bar{x}, \bar{a})\Big)^{-1} \frac{\partial(\nabla_x f(\bar{x}, \bar{a}))}{\partial a}. \tag{5.7}$$

Moreover, each such $x^*(a)$ with $a \in \mathcal{N}$ satisfies again the SOSC conditions and is thus a strict local minimizer.

Proof. The existence of the differentiable map $x^* : \mathcal{N} \to D$ follows from the implicit function theorem applied to the stationarity condition $\nabla_x f(x^*(a), a) = 0$. We recall the derivation of Eq. (5.7) via

$$0 = \frac{d(\nabla_x f(x^*(a), a))}{da} = \underbrace{\frac{\partial(\nabla_x f(x^*(a), a))}{\partial x}}_{=\nabla_x^2 f} \cdot \frac{\partial x^*(a)}{\partial a} + \frac{\partial(\nabla_x f(x^*(a), a))}{\partial a}$$

The fact that all points $x^*(a)$ satisfy the SOSC conditions follows from continuity of the second derivative. □

Chapter 6

Estimation and Fitting Problems

Estimation and fitting problems are optimization problems with a special objective, namely a "least squares objective"[1],

$$\min_{x \in \mathbb{R}^n} \ \frac{1}{2}\|\eta - M(x)\|_2^2. \tag{6.1}$$

Here, $\eta \in \mathbb{R}^m$ are the m "measurements" and $M : \mathbb{R}^n \to \mathbb{R}^m$ is a "model", and $x \in \mathbb{R}^n$ are called "model parameters". If the true value for x would be known, we could evaluate the model $M(x)$ to obtain model predictions for the measurements. The computation of $M(x)$, which might be a very complex function and for example involve the solution of a differential equation, is sometimes called the "forward problem": for given model inputs, we determine the model outputs.

In estimation and fitting problems, as (6.1), the situation is inversed: we want to find those model parameters x that yield a prediction $M(x)$ that is as close as possible to the actual measurements η. This problem is often called an "inverse problem": for given model outputs η, we want to find the corresponding model inputs x.

This type of optimization problem arises in applications like:

- function approximation
- online estimation for process control
- weather forecast (weather data reconciliation)
- parameter estimation

[1]Definition [Euclidean norm]: For a vector $x \in \mathbb{R}^n$, we define the norm as $\|x\|_2 = \left(\sum_{i=1}^n x_i^2\right)^{1/2} = (x^T x)^{1/2}$.

6.1 Linear Least Squares

Many models in estimation and fitting problems are linear functions of x. If M is linear, $M(x) = Jx$, then $f(x) = \frac{1}{2} \parallel \eta - Jx \parallel_2^2$ which is a convex function, as $\nabla^2 f(x) = J^T J \succcurlyeq 0$. Therefore local minimizers are found by

$$\begin{aligned} \nabla f(x^*) = 0 &\Leftrightarrow J^T J x^* - J^T \eta = 0 \\ &\Leftrightarrow x^* = \underbrace{(J^T J)^{-1} J^T}_{=J^+} \eta \end{aligned} \tag{6.2}$$

Definition 6.1 (Pseudo-inverse)
J^+ is called the pseudo-inverse and is a generalization of the inverse matrix. If $J^T J \succ 0$, J^+ is given by

$$J^+ = (J^T J)^{-1} J^T \tag{6.3}$$

So far, $(J^T J)^{-1}$ is only defined when $J^T J \succ 0$. This holds if and only if $\text{rank}(J) = n$, i.e., if the columns of J are linearly independent.

Example 6.1 (Average linear least squares)**:** Let us regard the simple optimization problem:

$$\min_{x \in \mathbb{R}} \ \frac{1}{2} \sum_{i=1}^{m} (\eta_i - x)^2.$$

This is a linear least squares problem, where the vector η and the matrix $J \in \mathbb{R}^{m \times 1}$ are given by

$$\eta = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix}, \ J = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}. \tag{6.4}$$

Because $J^T J = m$, it can be easily seen that

$$J^+ = (J^T J)^{-1} J^T = \frac{1}{m} \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \tag{6.5}$$

so we conclude that the local minimizer equals the average $\hat{\eta}$ of the given points η_i:

$$x^* = J^+ \eta = \frac{1}{m} \sum_{i=1}^{m} \eta_i = \hat{\eta}. \tag{6.6}$$

Example 6.2 (Linear Regression)**:** Given data points $\{t_i\}_{i=1}^{i=m}$ with corresponding values $\{\eta_i\}_{i=1}^{i=m}$, find the 2-dimensional parameter vector $x = (x_1, x_2)$, so that the polynomial of degree one

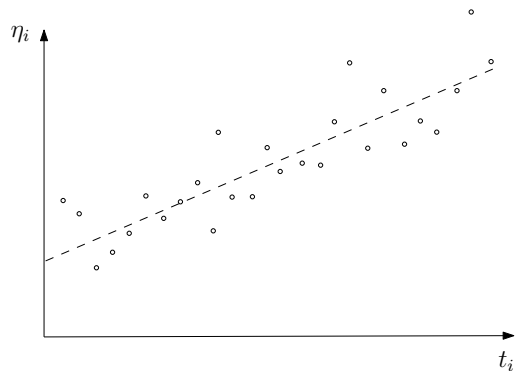


Figure 6.1: Linear regression for a set of data points (t_i, η_i)

$p(t; x) = x_1 + x_2 t$ provides a prediction of η at time t. The corresponding optimization problem looks like:

$$\min_{x \in \mathbb{R}^2} \frac{1}{2} \sum_{i=1}^{m} (\eta_i - p(t_i; x))^2 = \min_{x \in \mathbb{R}^2} \frac{1}{2} \left\| \eta - J \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right\|_2^2 \tag{6.7}$$

where η is the same vector as in (6.4) and J is given by

$$J = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_m \end{bmatrix}. \tag{6.8}$$

The local minimizer is found by equation (6.3), whereas the calculation of $(J^T J)$ is straightforward:

$$J^T J = \left[\begin{array}{c|c} m & \sum t_i \\ \hline \sum t_i & \sum t_i^2 \end{array} \right] = m \begin{bmatrix} 1 & \hat{t} \\ \hat{t} & \hat{t^2} \end{bmatrix} \tag{6.9}$$

In order to obtain x^*, first $(J^T J)^{-1}$ is calculated[2]:

$$(J^T J)^{-1} = \frac{1}{\det(J^T J)} \text{adj}(J^T J) = \frac{1}{m(\hat{t^2} - (\hat{t})^2)} \begin{bmatrix} \hat{t^2} & -\hat{t} \\ -\hat{t} & 1 \end{bmatrix}. \tag{6.10}$$

Second, we compute $J^T \eta$ as follows:

$$J^T \eta = \begin{bmatrix} 1 & \cdots & 1 \\ t_1 & \cdots & t_m \end{bmatrix} \begin{bmatrix} \eta_1 \\ \vdots \\ \eta_m \end{bmatrix} = \begin{bmatrix} \sum \eta_i \\ \sum \eta_i t_i \end{bmatrix} = m \begin{bmatrix} \hat{\eta} \\ \hat{\eta t} \end{bmatrix}. \tag{6.11}$$

Hence, the local minimizer is found by combining the expressions (6.10) and (6.11). Note that

$$\hat{t^2} - (\hat{t})^2 = \frac{1}{m} \sum (t_i - \hat{t})^2 = \sigma_t^2. \tag{6.12}$$

[2]Recall that the adjugate of a matrix $A \in \mathbb{R}^{nxn}$ is given by taking the transpose of the cofactor matrix, $\text{adj}(A) = C^T$ where $C_{ij} = (-1)^{i+j} M_{ij}$ with M_{ij} the (i, j) minor of A.

where we used in the last transformation a standard definition of the variance σ_t. The correlation coefficient ρ is similarly defined by

$$\rho = \frac{\sum(\eta_i - \hat{\eta})(t_i - \hat{t})}{m\sigma_t\sigma_\eta} = \frac{\hat{t\eta} - \hat{\eta}\hat{t}}{\sigma_t\sigma_\eta}. \tag{6.13}$$

The two-dimensional parameter vector $x = (x_1, x_2)$ is found:

$$x^* = \frac{1}{\sigma_t^2}\begin{bmatrix}\hat{t^2}\hat{\eta} - \hat{t}\hat{\eta}\hat{t} \\ -\hat{t}\hat{\eta} + \hat{\eta}\hat{t}\end{bmatrix} = \begin{bmatrix}\hat{\eta} - \hat{t}\frac{\sigma_\eta}{\sigma_t}\rho \\ \frac{\sigma_\eta}{\sigma_t}\rho\end{bmatrix}. \tag{6.14}$$

Finally, this can be written as a polynomial of first degree:

$$p(t; x^*) = \hat{\eta} + (t - \hat{t})\frac{\sigma_\eta}{\sigma_t}\rho. \tag{6.15}$$

6.2 Ill Posed Linear Least Squares

Definition (6.3) of the pseudo-inverse holds only when J^TJ is invertible, which implies that the set of optimal solutions S^* has only one optimal point x^*, given by equation (6.3): $S^* = \{x^*\} = (J^TJ)^{-1}J\eta$. If J^TJ is not invertible, the set of solutions S^* is given by

$$S^* = \{x \mid \nabla f(x) = 0\} = \{x | J^TJx - J^T\eta = 0\} \tag{6.16}$$

In order to pick a unique point out of this set, we might choose to search for the "minimum norm solution", i.e. the vector x^* with minimum norm satisfying $x^* \in S^*$.

$$\min_{x\in\mathbb{R}^n} \quad \frac{1}{2}\|x\|_2^2 \quad \text{subject to} \quad x \in S^* \tag{6.17}$$

We will show below that this minimal norm solution is given by the so called "Moore Penrose Pseudo Inverse".

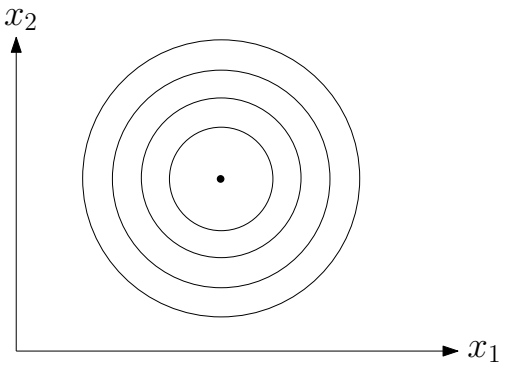


Figure 6.2: J^TJ is invertible, resulting in a unique minimum.

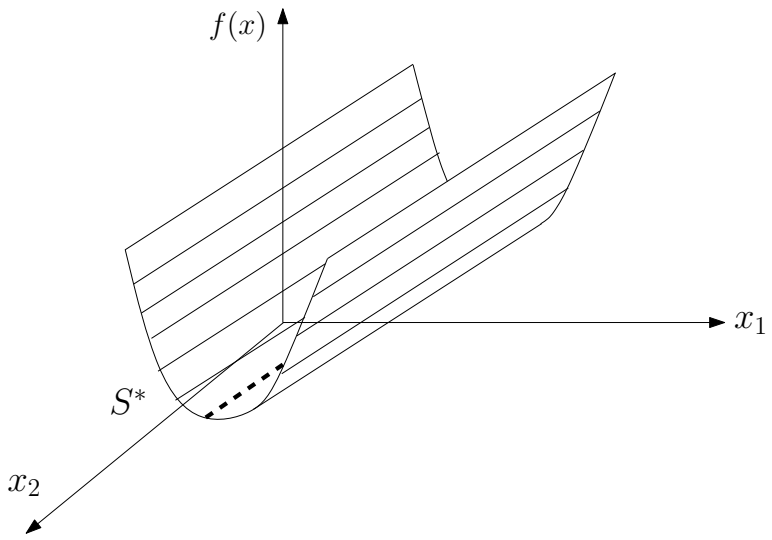


Figure 6.3: An example of an ill-posed problem, $J^T J$ is not invertible

Definition 6.2 (Moore Penrose Pseudo Inverse)
Assume $J \in \mathbb{R}^{m \times n}$ and that the singular value decomposition (SVD) of J is given by $J = USV^T$. Then, the Moore Penrose pseudo inverse J^+ is given by:

$$J^+ = VS^+U^T, \tag{6.18}$$

where for

$$S = \begin{bmatrix} \sigma_1 & & & & & & \\ & \sigma_2 & & & & & \\ & & \ddots & & & & \\ & & & \sigma_r & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 0 \\ \hline 0 & & \dots & 0 & \dots & & 0 \end{bmatrix} \quad \text{holds} \quad S^+ = \left[\begin{array}{ccccccc|c} \sigma_1^{-1} & & & & & & & 0 \\ & \sigma_2^{-1} & & & & & & \vdots \\ & & \ddots & & & & & 0 \\ & & & \sigma_r^{-1} & & & & \vdots \\ & & & & 0 & & & \\ & & & & & \ddots & & \\ & & & & & & 0 & 0 \end{array}\right] \tag{6.19}$$

If $J^T J$ is invertible, then $J^+ = (J^T J)^{-1} J^T$ what easily can be shown:

$$\begin{aligned} (J^T J)^{-1} J^T &= (VS^T U^T USV^T)^{-1} VS^T U^T \\ &= V(S^T S)^{-1} V^T VS^T U^T \\ &= V(S^T S)^{-1} S^T U^T \\ &= V \begin{bmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_r^2 \end{bmatrix}^{-1} \left[\begin{array}{ccc|c} \sigma_1 & & & \\ & \sigma_2 & & 0 \\ & & \ddots & \\ & & \sigma_r & \end{array}\right] U^T \\ &= VS^+U^T \end{aligned}$$

6.3 Regularization for Least Squares

The minimum norm solution can be approximated by a "regularized problem"

$$\min_x \ \frac{1}{2}\|\eta - Jx\|_2^2 + \frac{\epsilon}{2}\|x\|_2^2, \tag{6.20}$$

with small $\epsilon > 0$, to get a unique solution

$$\begin{aligned} \nabla f(x) &= J^T Jx - J^T\eta + \epsilon x && (6.21)\\ &= (J^T J + \epsilon\mathbb{I})x - J^T\eta && (6.22)\\ x^* &= (J^T J + \epsilon\mathbb{I})^{-1} J^T\eta && (6.23)\\ & && (6.24) \end{aligned}$$

Lemma 6.1: *for $\epsilon \to 0$, $(J^T J + \epsilon\mathbb{I})^{-1} J^T \to J^+$, with J^+ the Moore Penrose inverse.*

Proof. Taking the SVD of $J = USV^T$, $(J^T J + \epsilon\mathbb{I})^{-1} J^T$ can be written in the form:

$$\begin{aligned} (J^T J + \epsilon\mathbb{I})^{-1} J^T &= (VS^T U^T USV^T + \epsilon \underbrace{\mathbb{I}}_{VV^T})^{-1} \underbrace{J^T}_{US^T V^T}\\ &= V(S^T S + \epsilon\mathbb{I})^{-1} V^T VS^T U^T\\ &= V(S^T S + \epsilon\mathbb{I})^{-1} S^T U^T \end{aligned}$$

Rewriting the right hand side of the equation explicitly:

$$= V \begin{bmatrix} \sigma_1^2+\epsilon & & & & & \\ & \ddots & & & & \\ & & \sigma_r^2+\epsilon & & & \\ & & & \epsilon & & \\ & & & & \ddots & \\ & & & & & \epsilon \end{bmatrix}^{-1} \left[\begin{array}{cccc|c} \sigma_1 & & & & 0 \\ & \ddots & & & \vdots \\ & & \sigma_r & & \\ & & & 0 & \\ & & & & \ddots \\ & & & & 0 \end{array}\right] U^T$$

Calculating the matrix product simplifies the equation:

$$= V \left[\begin{array}{cccccc|c} \frac{\sigma_1}{\sigma_1^2+\epsilon} & & & & & & 0 \\ & \ddots & & & & & \\ & & \frac{\sigma_r}{\sigma_r^2+\epsilon} & & & & \vdots \\ & & & \frac{0}{\epsilon} & & & \\ & & & & \ddots & & \\ & & & & & \frac{0}{\epsilon} & 0 \end{array}\right] U^T$$

It can be easily seen that for $\epsilon \to 0$ each diagonal element has the solution:

$$\lim_{\epsilon\to 0} \frac{\sigma_i}{\sigma_i^2+\epsilon} = \begin{cases} \frac{1}{\sigma_i} & \text{if} \quad \sigma_i \neq 0 \\ 0 & \text{if} \quad \sigma_i = 0 \end{cases} \tag{6.25}$$

□

We have shown that the Moore Penrose inverse J^+ solves the problem (6.20) for infinitely small $\epsilon > 0$. Thus it selects $x^* \in S^*$ with minimal norm.

6.4 Statistical Derivation of Least Squares

A least squares problem (6.1) can be interpreted as finding the x that "explains" the noisy measurements η "best".

Definition 6.3 (Maximum-Likelihood Estimate)
A maximum-likelihood estimate of the unknown parameter x maximizes the probability $P(\eta|x)$ of obtaining the (given) measurements η if the parameter would have the value x.

Assume $\eta_i = M_i(\overline{x}) + \epsilon_i$ with $\overline{x}$ the "true" parameter, and ϵ_i *Gaussian noise* with expectation value $\mathbb{E}(\epsilon_i) = 0$, $\mathbb{E}(\epsilon_i\,\epsilon_i) = \sigma_i^2$ and ϵ_i, ϵ_j independent. Then holds

$$P(\eta|x) = \prod_{i=1}^{m} P(\eta_i \mid x) \tag{6.26}$$

$$= C \prod_{i=1}^{m} \exp\left(\frac{-(\eta_i - M_i(x))^2}{2\sigma_i^2}\right) \tag{6.27}$$

with $C = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma_i^2}}$. Taking the logarithm of both sides gives

$$\log P(\eta|x) = \log(C) + \sum_{i=1}^{m} -\frac{(\eta_i - M_i(x))^2}{2\sigma_i^2} \tag{6.28}$$

with a constant C. Due to monotonicity of the logarithm holds that the argument maximizing $P(\eta|x)$ is given by

$$\arg\max_{x\in\mathbb{R}^n} P(\eta|x) = \arg\min_{x\in\mathbb{R}^n} \; -\log(P(\eta|x)) \tag{6.29}$$

$$= \arg\min_{x\in\mathbb{R}^n} \; \sum_{i=1}^{m} \frac{(\eta_i - M_i(x))^2}{2\sigma_i^2} \tag{6.30}$$

$$= \arg\min_{x\in\mathbb{R}^n} \; \frac{1}{2} \left\| S^{-1}(\eta - M(x)) \right\|_2^2 \tag{6.31}$$

Thus, the least squares problem has a statistical interpretation. Note that due to the fact that we might have different standard deviations σ_i for different measurements η_i we need to scale both measurements and model functions in order to obtain an objective in the usual least squares form

$\|\hat{\eta} - \hat{M}(x)\|_2^2$, as

$$\min_x \ \frac{1}{2} \sum_{i=1}^{n} \left(\frac{\eta_i - M_i(x)}{\sigma_i} \right)^2 = \min_x \ \frac{1}{2} \|S^{-1}(\eta - M(x))\|_2^2 \tag{6.32}$$

$$= \min_x \ \frac{1}{2} \|S^{-1}\eta - S^{-1}M(x)\|_2^2 \tag{6.33}$$

$$\text{with } S = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_m \end{bmatrix}.$$

Statistical Interpretation of Regularization terms: Note that a regularization term like $\alpha\|x - \bar{x}\|_2^2$ that is added to the objective can be interpreted as a "pseudo measurement" $\bar{x}$ of the parameter value x, which includes a statistical assumption: the smaller α, the larger we implicitly assume the standard deviation of this pseudo-measurement. As the data of a regularization term are usually given before the actual measurements, regularization is also often interpreted as "a priori knowledge". Note that not only the Euclidean norm with one scalar weighting α can be chosen, but many other forms of regularization are possible, e.g. terms of the form $\|A(x - \bar{x})\|_2^2$ with some matrix A.

6.5 L1-Estimation

Instead of using $\|.\|_2^2$, i.e. the L2-norm in equation (6.1), we might alternatively use $\|.\|_1$, i.e., the L1-norm. This gives rise to the so called L1-estimation problem:

$$\min_x \|\eta - M(x)\|_1 = \min_x \ \sum_{i=1}^{m} |\eta_i - M_i(x)| \tag{6.34}$$

Like the L2-estimation problem, also the L1-estimation problem can be interpreted statistically as a maximum-likelihood estimate. However, in the L1-case, the measurement errors are assumed to follow a Laplace distribution instead of a Gaussian.

An interesting observation is that the optimal L1-fit of a constant x to a sample of different scalar values $\eta_1, \ldots, \eta_m$ just gives the median of this sample, i.e.

$$\arg\min_{x \in \mathbb{R}} \ \sum_{i=1}^{m} |\eta_i - x| \quad = \quad \text{median of } \{\eta_1, \ldots, \eta_m\}. \tag{6.35}$$

Remember that the same problem with the L2-norm gave the average of η_i. Generally speaking, the median is less sensitive to outliers than the average, and a detailed analysis shows that the solution to general L1-estimation problems is also less sensitive to a few outliers. Therefore, L1-estimation is sometimes also called "robust" parameter estimation.

6.6 Gauss-Newton (GN) Method

Linear least squares problems can be solved easily. Solving non-linear least squares problems globally is in general difficult, but in order to find a local minimum we can iteratively solve it, and in each iteration approximate the problem by its linearization at the current guess. This way we obtain a better guess for the next iterate, etc., just as in Newton's method for root finding problems.

For non-linear least squares problems of the form

$$\min_x \underbrace{\frac{1}{2}\|\eta - M(x)\|_2^2}_{=f(x)} \tag{6.36}$$

the so called "Gauss-Newton (GN) method" is used. To describe this method, let us first for notational convenience introduce the shorthand $F(x) = \eta - M(x)$ and redefine the objective to

$$f(x) = \frac{1}{2}\|F(x)\|_2^2 \tag{6.37}$$

where $F(x)$ is a nonlinear function $F : \mathbb{R}^n \to \mathbb{R}^m$ with $m > n$ (more measurements than parameters). At a given point x_k (iterate k), $F(x)$ is linearized, and the next iterate x_{k+1} obtained by solving a linear least squares problem. We expand

$$F(x) \approxeq F(x_k) + J(x_k)(x - x_k) \tag{6.38}$$

where $J(x)$ is the Jacobian of $F(x)$ which is defined as

$$J(x) = \frac{\partial F(x)}{\partial x}. \tag{6.39}$$

Then, x_{k+1} can be found as solution of the following linear least squares problem:

$$x_{k+1} = \arg\min_x \frac{1}{2}\|F(x_k) + J(x_k)(x - x_k)\|_2^2 \tag{6.40}$$

For simplicity, we write $J(x_k)$ as J and $F(x_k)$ as F:

$$\begin{aligned} x_{k+1} &= \arg\min_x \frac{1}{2}\|F + J(x - x_k)\|_2^2 && (6.41)\\ &= x_k + \arg\min_p \frac{1}{2}\|F + Jp\|_2^2 && (6.42)\\ &= x_k - (J^T J)^{-1} J^T F && (6.43)\\ &= x_k + p_k^{\mathrm{GN}} && (6.44) \end{aligned}$$

In the next chapter the convergence theory of this method is treated, i.e., the question if the method converges, and to which point. The Gauss-Newton method is only applicable to least-squares problems, because the method linearizes the non-linear function inside the L2-norm. Note that in equation (6.43) $J^T J$ might not always be invertible.

6.7 Levenberg-Marquardt (LM) Method

This method is a generalization of the Gauss-Newton method that is in particular applicable if $J^T J$ is not invertible, and can lead to more robust convergence far from a solution. The Levenberg-Marquardt (LM) method makes the step p_k smaller by penalizing the norm of the step. It defines the step as:

$$p_k^{\mathrm{LM}} = \arg\min_p \frac{1}{2}\|F(x_k) + J(x_k)p\|_2^2 + \frac{\alpha_k}{2}\|p\|_2^2 \tag{6.45}$$

$$= -(J^T J + \alpha_k \mathbb{I})^{-1} J^T F \tag{6.46}$$

with some $\alpha_k > 0$. Using this step, it iterates as usual

$$x_{k+1} = x_k + p_k^{\mathrm{LM}}. \tag{6.47}$$

If we would make α_k very big, we would not correct the point, but we would stay where we are: for $\alpha_k \to \infty$ we get $p_k^{\mathrm{LM}} \to 0$. More precisely, $p_k^{\mathrm{LM}} = \frac{1}{\alpha_k} J^T F + O\left(\frac{1}{\alpha_k^2}\right)$. On the other hand, for small α_k, i.e. for $\alpha_k \to 0$ we get $p_k^{\mathrm{LM}} \to -J^+ F$.

It is interesting to note that the gradient of the least squares objective function $f(x) = \frac{1}{2}\|F(x)\|_2^2$ equals

$$\nabla f(x) = J(x)^T F(x), \tag{6.48}$$

which is the rightmost term in the step of both the Gauss-Newton and the Levenberg-Marquardt method. Thus, if the gradient equals zero, then also $p_k^{\mathrm{GN}} = p_k^{\mathrm{LM}} = 0$. This is a necessary condition for convergence to stationary points: the GN and LM method both stay at a point x_k with $\nabla f(x_k) = 0$. In the following chapter the convergence properties of these two methods will be analysed in much more detail. In fact, these two methods are part of a larger family, namely the "Newton type optimization methods".

Chapter 7

Newton Type Optimization

In this chapter we will treat how to solve a general unconstrained nonlinear optimization problem:

$$\min_{x \in \mathbb{R}^n} \quad f(x) \tag{7.1}$$

with $f \in C^2$

Definition 7.1 (Iterative Algorithm)
An "iterative algorithm" generates a sequence $x_0, x_1, x_2, ...$ of so called "iterates" with $x_k \to x^*$

7.1 Exact Newton's Method

In numerical analysis, Newton's method (or the Newton-Raphson method) is a method for finding roots of equations in one or more dimensions. Regard the equation:

$$\nabla f(x^*) \quad = \quad 0 \tag{7.2}$$

with $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$, which has as many components as variables.

The Newton idea consists of linearizing the non-linear equations at x_k to find $x_{k+1} = x_k + p_k$

$$\nabla f(x_k) + \underbrace{\frac{\partial}{\partial x}(\nabla f(x_k))}_{\nabla^2 f(x_k)} p_k \quad = \quad 0 \tag{7.3}$$

$$\Updownarrow$$

$$-\nabla^2 f(x_k)^{-1} \nabla f(x_k) \quad = \quad p_k \tag{7.4}$$

p_k is called the "Newton-step", $\nabla^2 f(x_k)$ is the Hessian.

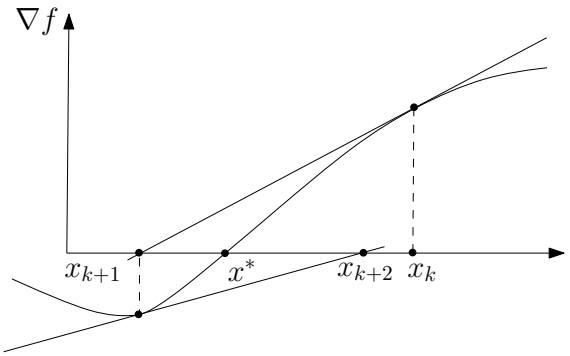


Figure 7.1: Visualization of the exact Newton's method.

A second interpretation of Newton's method for optimization can be obtained by a quadratic objective function, i.e. a second order Taylor approximation (a quadratic model can easily solved). The quadratic model m_k of objective f

$$m_k(x_k + p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p \tag{7.5}$$

$$\approxeq f(x_k + p) \tag{7.6}$$

There we would obtain step p_k by minimizing $m_k(x_k + p)$:

$$p_k = \arg\min_p m_k(x_k + p) \tag{7.7}$$

This is translated to the following equation that the optimal p must satisfy:

$$\nabla m(x_k + p) = \nabla f(x_k) + \nabla^2 f(x_k) p = 0 \tag{7.8}$$

Written explicitly for p_k

$$p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k) \tag{7.9}$$

which is the same formula, but with a different interpretation.

7.2 Local Convergence Rates

We will in a later section prove within a more general theorem that Newton's method converges quadratically if it is started close to a solution. For completeness, let's formulate this result already in this section, and define rigorously what "quadratic convergence" means.

Theorem 7.1 (Quadratic convergence of Newton's method)**:** *Suppose $f \in C^2$ and moreover,*

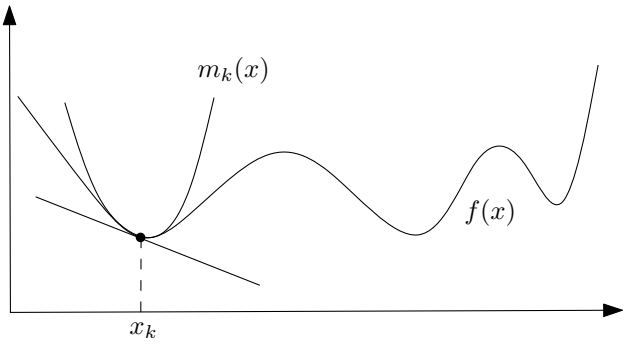


Figure 7.2: The second interpretation of Newton's method.

$\nabla^2 f(x)$ is a Lipschitz function[1] in a neighborhood of x^*. x^* is a local minimum satisfying SOSC ($\nabla f(x^*) = 0$ and $\nabla^2 f(x^*) \succ 0$). If x_0 is sufficiently close to x^*, then the Newton iteration $x_0, x_1, x_2, ...$

* *converges to x^*,*
* *converges with q-quadratic rate, and*
* *the sequence of $\|\nabla f(x_k)\|$ converges to zero quadratically.*

Proof. We refer to Theorem 3.5 from [4] and to our more general result in a later section of this chapter. □

Definition 7.2 (Different types of convergence rates)
Assume $x_k \in \mathbb{R}^n$, $x_k \to \overline{x}$. Then the sequence x_k is said to converge:

i. *Q-linearly* ⇔

$$\|x_{k+1} - \overline{x}\| \leqslant C\|x_k - \overline{x}\| \text{ with } C < 1 \tag{7.10}$$

holds for all $k \geqslant k_0$. The "Q" in Q-linearly means the "Q" of "quotient". Another equivalent definition is:

$$\limsup_{k\to\infty} \frac{\|x_{k+1} - \overline{x}\|}{\|x_k - \overline{x}\|} < 1 \tag{7.11}$$

ii. *Q-superlinearly* ⇔

$$\|x_{k+1} - \overline{x}\| \leqslant C_k\|x_k - \overline{x}\| \text{ with } C_k \to 0 \tag{7.12}$$

This is equivalent to:

$$\limsup_{k\to\infty} \frac{\|x_{k+1} - \overline{x}\|}{\|x_k - \overline{x}\|} = 0 \tag{7.13}$$

[1] A function f is a Lipschitz function if $\|f(x) - f(y)\| \leqslant L\|x - y\|$ for all x and y, where L is a constant independent of x and y.

iii. *Q-quadratically* $\Leftrightarrow$

$$\|x_{k+1} - \overline{x}\| \leqslant C\|x_k - \overline{x}\|^2 \text{ with } C < \infty \tag{7.14}$$

which is equivalent to:

$$\limsup_{k\to\infty} \frac{\|x_{k+1} - \overline{x}\|}{\|x_k - \overline{x}\|^2} < \infty \tag{7.15}$$

Example 7.1 (Convergence rates)**:** Consider examples with $x_k \in \mathbb{R}$, $x_k \to 0$ and $\overline{x} = 0$.

a) $x_k = \frac{1}{2^k}$ converges q-linearly: $\frac{x_{k+1}}{x_k} = \frac{1}{2}$.

b) $x_k = 0.99^k$ also converges q-linearly: $\frac{x_{k+1}}{x_k} = 0.99$. This example converges very slowly to $\overline{x}$. In practice we desire C in equation (7.10) be smaller than, say, $\frac{1}{2}$.

c) $x_k = \frac{1}{k!}$ converges Q-superlinearly, as $\frac{x_{k+1}}{x_k} = \frac{1}{k+1}$

d) $x_k = \frac{1}{2^{2^k}}$ converges Q-quadratically, because $\frac{x_{k+1}}{(x_k)^2} = \frac{(2^{2^k})^2}{2^{2^{k+1}}} = 1 < \infty$. For $k = 6$, $x^k = \frac{1}{2^{64}} \approx 0$, so in practice convergence up to machine precision is reached after roughly 6 iterations.

Definition 7.3 (R-convergence)
If the norm sequence $\|x_k - \overline{x}\|$ is upper bounded by some sequence $y_k \to 0, y_k \in \mathbb{R}$ i.e. $\|x_k - \overline{x}\| \leq y_k$ and if y_k is converging with a given Q-rate, i.e. Q-linearly, Q-superlinearly or Q-quadratically, then x_k is said to converge "R-linearly, R-superlinearly, or R-quadratically" to $\bar{x}$. Here, R indicates "root", because, e.g., R-linear convergence can also be defined via the root criterion $\lim_{k\to\infty} \sqrt[k]{\|x_k - \overline{x}\|} < 1$.

Example 7.2 (R-convergence)**:**

$$x_k = \begin{cases} \frac{1}{2^k} & \text{if k even} \\ 0 & \text{else} \end{cases} \tag{7.16}$$

This is a fast R-linear convergence, but it is not monotonically decreasing like Q-linear convergence.

Summary The three different Q-convergence and three different R-convergence rates have the following relations with each other. Here, $X \Rightarrow Y$ should be read as "If a sequence converges with rate X this implies that the sequence also converges with rate Y".

$$\begin{array}{ccccc} Q-quadratically & \Rightarrow & Q-superlinearly & \Rightarrow & Q-linearly \\ \Downarrow & & \Downarrow & & \Downarrow \\ R-quadratically & \Rightarrow & R-superlinearly & \Rightarrow & R-linearly \end{array}$$

7.3 Newton Type Methods

Any iteration of form

$$x_{k+1} = x_k - B_k^{-1}\nabla f(x_k) \tag{7.17}$$

with B_k invertible is called a "Newton type iteration for optimization". For $B_k = \nabla^2 f(x_k)$ we recover Newton's method, usually we try $B_k \approx \nabla^2 f(x_k)$. In each iteration, a quadratic model is minimized to obtain the next step, p_k:

$$p_k = \arg\min_p m_k(x_k + p) \tag{7.18}$$

The corresponding model is written in the form

$$m_k(x_k + p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T B_k p \tag{7.19}$$

This model leads to the step of Newton type iteration:

$$0 = \nabla m_k(x_k + p_k) = B_k p_k + \nabla f \tag{7.20}$$

$$\Leftrightarrow p_k = -B_k^{-1}\nabla f \tag{7.21}$$

Note that p_k is a minimizer of $m_k(x_k + p)$ only if $B_k \succ 0$. For exact Newton, this might not be the case for x_k far from the solution x^*.

Lemma 7.2 (Descent direction)**:** *If $B_k \succ 0$ then $p_k = -B_k^{-1}\nabla f(x_k)$ is a descent direction.*

Proof.

$$\nabla f(x_k)^T p_k = \underbrace{-\nabla f(x_k)^T \underbrace{B_k^{-1}}_{\succ 0} \nabla f(x_k)}_{>0} < 0 \tag{7.22}$$

□

In the next part of this section, we consider two questions:

1. Can we guarantee convergence for any initial guess x_0? The answer can be found in the chapter on "global convergence".
2. How fast is the "local convergence" rate? We will first approach this question by a few examples.

Definition 7.4 (Newton type variants)
This section discusses some Newton type variants, frequently used:

a) Gauss-Newton and Levenberg-Marquardt: for $f(x) = \frac{1}{2}\|F(x)\|_2^2$ take

$$\begin{aligned} m_k(x_k + p) &= \frac{1}{2}\|F(x_k) + J(x_k)p\|_2^2 + \frac{\alpha_k}{2}\|p\|_2^2 \quad (7.23) \\ &= \frac{1}{2}\|F(x_k)\|_2^2 + p^T J(x_k)^T F(x_k) + \frac{1}{2}p^T(J(x_k)^T J(x_k) + \alpha_k \mathbb{I})p \end{aligned}$$

where $J(x_k)^T F(x_k)$ equals the gradient, $\nabla f(x_k)$ of f. In the Gauss-Newton and Levenberg-Marquardt method, we have

$$B_k = J(x_k)^T J(x_k) + \alpha_k \mathbb{I} \quad (7.24)$$

and step $p_k = -B_k^{-1}\nabla f(x_k)$. When is B_k close to $\nabla^2 f(x_k)$? Note that

$$F(x) = \begin{bmatrix} F_1(x) \\ F_2(x) \\ \vdots \\ F_m(x) \end{bmatrix} \quad (7.25)$$

The Hessian $\nabla^2 f(x_k)$ is then computed as:

$$\begin{aligned} \nabla^2 f(x) &= \frac{\partial}{\partial x}(\nabla f(x)) = \frac{\partial}{\partial x}(J(x)^T F(x)) \quad (7.26) \\ &= \frac{\partial}{\partial x}(\sum_{i=1}^{m} \nabla F_i(x) F_i(x)) \quad (7.27) \\ &= J(x)^T J(x) + \sum_{i=1}^{m} \nabla^2 F_i(x) F_i(x) \quad (7.28) \end{aligned}$$

In Gauss-Newton, we have $\nabla^2 f(x) - B_k = \sum_{i=1}^{m} \nabla^2 F_i(x) F_i(x)$. This "error matrix" gets small if

* $\nabla^2 F_i(x)$ are small (F nearly linear)
* $F_i(x)$ are small $\Leftrightarrow$ "good fit" or "small residuals"

Gauss Newton works well for small residual problems. If you have a solution with perfect fit, a locally quadratic convergence rate is reached at the end of the iterates.

b) Steepest descent method or gradient method: Take $B_k = \alpha_k \mathbb{I}$ and

$$p_k = -B_k^{-1}\nabla f(x_k) = -\frac{\nabla f(x_k)}{\alpha_k} \quad (7.29)$$

This is the negative gradient, the direction of steepest descent. But how to choose α_k, or equivalently, how long to take steps? "Line search" as explained later, will be one answer to this.

c) Quasi-Newton methods: Approximate Hessian B_{k+1} from knowledge of B_k and $\nabla f(x_k)$ and $\nabla f(x_{k+1})$. We get the following important equation:

$$B_{k+1}(x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k), \quad (7.30)$$

the so called "secant condition".
As an example, consider the BFGS-formula:

$$B_{k+1} = B_k - \frac{B_k s s^T B_k}{s^T B_k s} + \frac{y y^T}{s^T y} \tag{7.31}$$

with s and y defined as:

$$s = x_{k+1} - x_k, \tag{7.32}$$
$$y = \nabla f(x_{k+1}) - \nabla f(x_k). \tag{7.33}$$

We easily check that $B_{k+1}s = y$. The BFGS method is a very successful method, and it can be shown that $B_k \to \nabla^2 f(x^*)$.

d) Inexact Newton: Solve the linear system

$$\nabla^2 f(x_k)p = -\nabla f(x_k) \tag{7.34}$$

inexactly, e.g. by iterative linear algebra. This approach is good for large scale problems.

7.4 Local Convergence for Newton Type Methods

Theorem 7.3 (Local Contraction for Newton Type Methods)**:** *Assume x^* satisfies SOSC for $f \in C^2$. We regard Newton type iteration $x_{k+1} = x_k + p_k$, where p_k is given by*

$$p_k = -B_k^{-1}\nabla f(x_k) \tag{7.35}$$

with B_k invertible $\forall k \in \mathbb{N}$.
We assume a Lipschitz condition on the Hessian $\nabla^2 f$:

$$\|B_k^{-1}(\nabla^2 f(x_k) - \nabla^2 f(y))\| \leq \omega\|x_k - y\| \tag{7.36}$$

that holds for $\forall k \in \mathbb{N}$, $y \in \mathbb{R}^n$, with $\omega < \infty$ a Lipschitz constant. We also assume a compatibility condition

$$\|B_k^{-1}(\nabla^2 f(x_k) - B_k)\| \leq \kappa_k \quad \forall k \in \mathbb{N} \tag{7.37}$$

with $\kappa_k \leq \kappa$ and $\kappa < 1$. We also assume that

$$\|x_0 - x^*\| < \frac{2(1-\kappa)}{\omega} \tag{7.38}$$

Then $x_k \to x^*$ and

i) If $\kappa = 0$ (Exact Newton) then the rate is Q-quadratic

ii) If $\kappa_k \to 0$ (Quasi Newton method) then the rate is Q-superlinear

iii) If $\kappa > 0$, $\kappa_k > \rho > 0$ (Gauss-Newton, Levenberg-Marquardt, steepest descent) then the rate is Q-linear.

Proof. We will show that $\|x_{k+1} - x^*\| \leq \delta_k \|x_k - x^*\|$ with $\delta_k < 1$. For this aim let us regard

$$\begin{aligned} x_{k+1} - x^* &= x_k - x^* - B_k^{-1} \nabla f(x_k) \\ &= x_k - x^* - B_k^{-1}(\nabla f(x_k) - \nabla f(x^*)) \\ &= B_k^{-1}(B_k(x_k - x^*)) - B_k^{-1} \int_0^1 \nabla^2 f(x^* + t(x_k - x^*))(x_k - x^*) dt \\ &= B_k^{-1}(B_k - \nabla^2 f(x_k))(x_k - x^*) - B_k^{-1} \int_0^1 \Big[\nabla^2 f(x^* + t(x_k - x^*)) - \nabla^2 f(x_k)\Big](x_k - x^*) dt \end{aligned}$$

Taking the L2-norm of both sides:

$$\begin{aligned} \|x_{k+1} - x^*\| &\leq \kappa_k \|x_k - x^*\| + \int_0^1 \omega \|x^* + t(x_k - x^*) - x_k\| dt \quad \|x_k - x^*\| \\ &= \Big(\kappa_k + \omega \underbrace{\int_0^1 (1-t) dt}_{=\frac{1}{2}} \|x_k - x^*\|\Big) \|x_k - x^*\| \\ &= \underbrace{\Big(\kappa_k + \frac{\omega}{2} \|x_k - x^*\|\Big)}_{=\delta_k} \|x_k - x^*\| \end{aligned}$$

We distinguish three different convergence rates depending on the value of κ respectively κ_k:

i) $\|x_{k+1} - x^*\| \leq \frac{\omega}{2} \|x_k - x^*\|^2$, Q-quadratic

ii) $\|x_{k+1} - x^*\| \leq \underbrace{(\kappa_k + \frac{\omega}{2} \|x_k - x^*\|)}_{\to 0} \|x_k - x^*\|$, Q-superlinear

iii) $\|x_{k+1} - x^*\| \leq (\underbrace{\kappa_k}_{<1} + \underbrace{\frac{\omega}{2} \|x_k - x^*\|}_{\to 0}) \|x_k - x^*\|$, Q-linear

□

Chapter 8

Globalisation Strategies

A Newton-type method only converges locally if

$$\kappa + \frac{\omega}{2}\|x_0 - x^*\| < 1 \tag{8.1}$$

$$\Updownarrow \tag{8.2}$$

$$\|x_0 - x^*\| < \frac{2(1-\kappa)}{\omega} \tag{8.3}$$

Recall that ω is a Lipschitz constant of the Hessian that is bounding the non-linearity of the problem, and κ is a measure of the approximation error of the Hessian. But what if $\|x_0 - x^*\|$ is too big to make Newton's method converge locally?

The general idea is to make the steps in the iteration shorter and to ensure descent: $f(x_{k+1}) < f(x_k)$. This shall result in $\nabla f(x_k) \to 0$. While doing this, we should not take too small steps and get stuck. In this chapter two methods will be described to solve this problem: *Line-search* and *Trust-region*.

8.1 Line-Search Method

Each iteration of a line search method computes first a search direction p_k. The idea is to require p_k to be a descent direction[1]. The iteration is then given by

$$x_{k+1} = x_k + t_k p_k \tag{8.4}$$

with $t_k \in (0, 1]$ a scalar called the *step length* ($t_k = 1$ in case of a full step Newton type method).

Computing the step length t_k requires a tradeoff between a substantial reduction of f and the computing speed of this minimization problem. Regard the ideal line search minimization:

$$\min_t f(x_k + tp_k) \quad \text{subject to} \quad t \in (0, 1] \tag{8.5}$$

[1] p_k is a descent direction iff $\nabla f(x_k)^T p_k < 0$

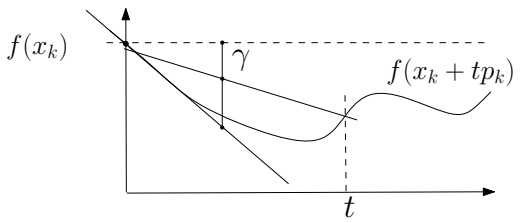


Figure 8.1: A visualization of the Armijo sufficient decrease condition.

Exact line search is not necessary, instead we ensure that (a) the steps are short enough to get sufficient decrease (descending must be relevant) and (b) long enough to not get stuck.

a) "Armijo's" sufficient decrease condition stipulates that t_k should give sufficient decrease in f:

$$f(x_k + t_k p_k) \leq f(x_k) + \gamma t_k \nabla f(x_k)^T p_k \tag{8.6}$$

with $\gamma \in (0, \frac{1}{2})$ the relaxation of the gradient. In practice γ is chosen quite small, say $\gamma = 0.1$ or even smaller. Note that with $\gamma = 1$, the right hand side of equation (8.6) would be a first order Taylor expansion.

This condition alone, however, only ensures that the steps are not too long, and it is not sufficient to ensure that the algorithm makes fast enough progress. Many ways exist to make sure that the steps do not get too short either, and we will just learn one of them.

b) Backtracking chooses the step length by starting with $t = 1$ and checking it against Armijo's condition. If the Armijo condition is not satisfied, t will be reduced by a factor $\beta \in (0, 1)$. In practice β is chosen to be not too small, e.g. $\beta = 0.8$.

A basic implementation of a) and b) can be found in Algorithm 1.

Algorithm 1 Backtracking with Armijo Condition

Inputs: x_k, p_k, $f(x_k)$, $\nabla f(x_k)^T p_k$, γ, β
Output: step length t_k

$t \leftarrow 1$
while $f(x_k + tp_k) \geq f(x_k) + \gamma t \nabla f(x_k)^T p_k$ **do**
  $t \leftarrow \beta t$
end while
$t_k \leftarrow t$

Example 8.1: (Armijo: not sufficient decrease) An example where no convergence is reached is shown in Figure 8.2. In this example we consider the function

$$f(x) = x^2 \tag{8.7}$$

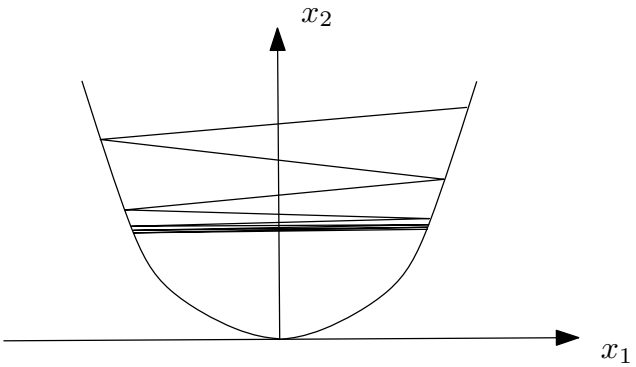


Figure 8.2: Visualization of example 8.1, illustrating that even if $f(x_{k+1}) < f(x_k)$ in each iteration, because of insufficient decrease, $x^* = 0$ is not reached. This behaviour would be excluded by the Armijo condition.

with $x_0 = -1$, $p_k = (-1)^k$ and t_k defined as

$$t_k = \left(2 - \left(\frac{1}{4}\right)^{k+1}\right) |x_k| \tag{8.8}$$

Remark that $f(x_{k+1}) = f(x_k + t_k p_k) < f(x_k)$ but no convergence to $x^* = 0$ is reached!

8.2 Global Convergence of the Line Search Method

We will now state a general algorithm for Newton type line search, Algorithm 2.

Algorithm 2 Newton type line search

Inputs: x_0, TOL > 0, $\beta \in (0, 1)$, $\gamma \in (0, \frac{1}{2})$
Output: x^*

$k \leftarrow 0$
while $\|\nabla f(x_k)\| >$ TOL **do**
  obtain $B_k \succ 0$
  $p_k \leftarrow -B_k^{-1} \nabla f(x_k)$
  get t_k from the backtracking algorithm
  $x_{k+1} \leftarrow x_k + t_k p_k$
  $k \leftarrow k + 1$
end while
$x^* \leftarrow x_k$

Note: For computational efficiency, $\nabla f(x_k)$ should only be evaluated once in each iteration.

Theorem 8.1 (Global Convergence of Line-Search)**:** *Assume $f \in C^1$ (once differentiable) with ∇f Lipschitz and $c_1\mathbb{I} \preceq B_k^{-1} \preceq c_2\mathbb{I}$ (eigenvalues of B_k^{-1}: $c_2 \geq \text{eig}(B_k^{-1}) \geq c_1$) with $0 < c_1 \ll c_2$. Then either Algorithm 2 stops with success, i.e., $\|\nabla f(x_k)\| \leq \text{TOL}$, or $f(x_k) \to -\infty$, i.e., the problem was unbounded below.*

Proof by contradiction. Assume that Algorithm 2 does not stop, i.e. $\|\nabla f(x_k)\| > \text{TOL}$ for all k, but that $f(x_k)$ is bounded below.

Because $f(x_{k+1}) \leq f(x_k)$ we have $f(x_k) \to f^*$ for some f^* which implies $[f(x_k) - f(x_{k+1})] \to 0$.

From Armijo (8.2), we have

$$\begin{aligned} f(x_k) - f(x_{k+1}) &\geq -\gamma t_k \nabla f(x_k)^T p_k && (8.9)\\ &= \gamma t_k \nabla f(x_k)^T B_k^{-1} \nabla f(x_k) && (8.10)\\ &\geq \gamma c_1 t_k \|\nabla f(x_k)\|_2^2 && (8.11) \end{aligned}$$

So we have already:

$$\gamma c_1 t_k \|\nabla f(x_k)\|_2^2 \to 0 \tag{8.12}$$

If we can show that $t_k \geq t_{\min} > 0, \forall k$ our contradiction is complete ($\Rightarrow \|\nabla f(x_k)\|_2^2 \to 0$).

We show that $t_k \geq t_{\min}$ with $t_{\min} = \min(1, \frac{(1-\gamma)\beta}{Lc_2}) > 0$ where L is the Lipschitz constant for ∇f, i.e., $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$.

For full steps $t_k = 1$, obviously $t_k \geq t_{\min}$. In the other case, due to backtracking, we must have for the *previous* line search step ($t = \frac{t_k}{\beta}$) that the Armijo condition is <u>not</u> satisfied, otherwise we would have accepted it.

$$\begin{aligned} f(x_k + \frac{t_k}{\beta} p_k) &> f(x_k) + \gamma \frac{t_k}{\beta} \nabla f(x_k)^T p_k && (8.13)\\ \Leftrightarrow \underbrace{f(x_k + \frac{t_k}{\beta} p_k) - f(x_k)}_{= \nabla f(x_k + \tau p_k)^T p_k \frac{t_k}{\beta}, \text{ for some } \tau \in (0, \frac{t_k}{\beta})} &> \gamma \frac{t_k}{\beta} \nabla f(x_k)^T p_k && (8.14)\\ \Leftrightarrow \nabla f(x_k + \tau p_k)^T p_k &> \gamma \nabla f(x_k)^T p_k && (8.15)\\ \Leftrightarrow \underbrace{(\nabla f(x_k + \tau p_k) - \nabla f(x_k))^T}_{\|\cdot\| \leq L\tau \|p_k\|} p_k &> (1-\gamma) \underbrace{(-\nabla f(x_k)^T p_k)}_{{p_k}^T B_k p_k} && (8.16)\\ \Rightarrow L \frac{t_k}{\beta} \|p_k\|^2 &> (1-\gamma) \underbrace{\|{p_k}^T B_k p_k\|}_{\geq \frac{1}{c_2}\|p_k\|_2^2} && (8.17)\\ \Rightarrow t_k &> \frac{(1-\gamma)\beta}{c_2 L} && (8.18) \end{aligned}$$

(Recall that $\frac{1}{c_1} \geq \text{eig}(B_k) \geq \frac{1}{c_2}$). We have shown that the step length will not be shorter than $\frac{(1-\gamma)\beta}{c_2 L}$, and will thus never become zero. □

8.3 Trust-Region Methods (TR)

"Line-search methods and trust-region methods both generate steps with the help of a quadratic model of the objective function, but they use this model in different ways. Line search methods use it to generate a search direction and then focus their efforts on finding a suitable step length α along this direction. Trust-region methods define a region around the current iterate within they trust the model to be an adequate representation of the objective function and then choose the step to be the approximate minimizer of the model in this region. In effect, they choose the direction and length of the step simultaneously. If a step is not acceptable, they reduce the size of the region and find a new minimzer. In general, the direction of the step changes whenever the size of the trust-region is altered. The size of the trust-region is critical to the effectiviness of each step." (cited from [4]).

The idea is to iterate $x_{k+1} = x_k + p_k$ with

$$p_k = \arg\min_{p \in \mathbb{R}^n} m_k(x_k + p) \text{ s.t. } \|p\| \leq \Delta_k \tag{8.19}$$

Equation (8.19) is called the TR-Subproblem, and $\Delta_k > 0$ is called the TR-Radius.

One particular advantage of this new type of subproblem is that we even can use indefinite Hessians without problems. Remember that – for an indefinite Hessian – the unconstrained quadratic model is not bounded below. A trust-region constraint will always ensure that the feasible set of the subproblem is bounded so that it always has a well-defined minimizer.

Before defining the "trustworthiness" of a model, recall that:

$$m_k(x_k + p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T B_k p \tag{8.20}$$

Definition 8.1 (Trustworthiness)
A measure for the *trustworthiness* of a model is the ratio of actual and predicted reduction.

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{\underbrace{m_k(x_k) - m_k(x_k + p_k)}_{>0 \text{ if } \|\nabla f(x_k)\| \neq 0}} = \frac{\mathrm{A}_{\mathrm{red}}}{\mathrm{P}_{\mathrm{red}}} \tag{8.21}$$

We have $f(x_k + p_k) < f(x_k)$ only if $\rho_k > 0$. $\rho_k \approx 1$ means a very trustworthy model. The trust-region algorithm is described in Algorithm 3.

The general convergence of the TR algorithm can be found in Theorem 4.5 in [4]

Algorithm 3 Trust-Region

Inputs: $\Delta_{\max}$, $\eta \in [0, \frac{1}{4}]$ *(when do we accept a step)*, Δ_0, x_0, TOL > 0
Output: x^*

$k = 0$
while $\|\nabla f(x_k)\| >$ TOL **do**
  Solve the TR-subproblem (8.19) (approximately) to get p_k
  Compute ρ_k
  Adapt Δ_{k+1}:
  if $\rho_k < \frac{1}{4}$ **then**
    $\Delta_{k+1} \leftarrow \Delta_k \cdot \frac{1}{4}$ *(bad model: reduce radius)*
  else if $\rho_k > \frac{3}{4}$ and $\|p_k\| = \Delta_k$ **then**
    $\Delta_{k+1} \leftarrow \min(2 \cdot \Delta_k, \Delta_{\max})$ *(good model: increase radius, but not too much)*
  else
    $\Delta_{k+1} \leftarrow \Delta_k$
  end if
  Decide on acceptance of step
  if $\rho_k > \eta$ **then**
    $x_{k+1} \leftarrow x_k + p_k$ *(we trust the model)*
  else
    $x_{k+1} \leftarrow x_k$ *"null" step*
  end if
  $k \leftarrow k + 1$
end while
$x^* \leftarrow x_k$

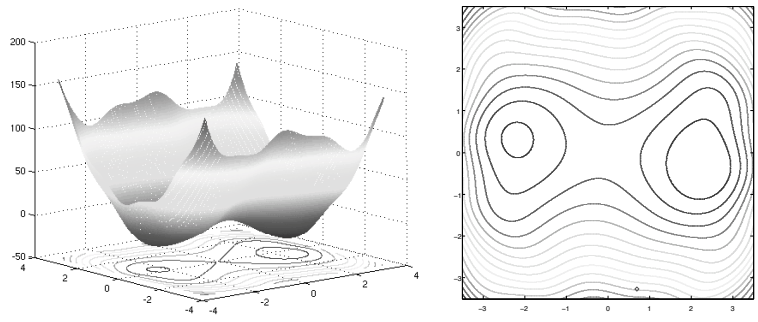


Figure 8.3: Visualisation of Example 8.2. The right image represents the contours of the objective, showing two local minima.

8.4 Trust-Region Example from Prof. Philippe Toint

This example was treated by Prof. Philippe Toint in the Francqui chair inaugural lecture at the KU Leuven, 20th April 2009 [2]. More information on the trust-region algorithm can be found in [2].

Example 8.2 (Trust-region algorithm example)**:** Consider the following objective to be mimimized

$$f(\alpha, \beta) \quad = \quad -10\alpha^2 + 10\beta^2 + 4\sin(\alpha\beta) - 2\alpha + \alpha^4 \tag{8.22}$$

This objective has two local minima: $(-2.20,\ 0.32)$ and $(2.30,\ -0.34)$.

The initial point or guess is $x_0 = (0.71, 3.27)$, where $f(x_0) = 97.630$. The contours of f and the contours of the quadratic model m_k around x_k are visualised in Figures 8.4-8.9. The trust-region iterates for this problem are summarized in Table 8.1.

k	Δ_k	s_k	$f(x_k + s_k)$	$\Delta f / \Delta m_k$	x_{k+1}
0	1	(0.05, 0.93)	43.742	0.998	$x_0 + s_0$
1	2	(-0.62, 1.78)	2.306	1.354	$x_1 + s_1$
2	4	(3.21, 0.00)	6.295	-0.004	x_2
3	2	(1.90, 0.08)	-29.392	0.649	$x_2 + s_2$
4	2	(0.32, 0.15)	-31.131	0.857	$x_3 + s_3$
5	4	(-0.03, -0.02)	-31.176	1.009	$x_4 + s_4$

Table 8.1: Summary of the trust-region algorithm iterates of Example 8.2

[2]The graphs here are shown with friendly permission of Philippe Toint and Annick Sartenaer who made them.

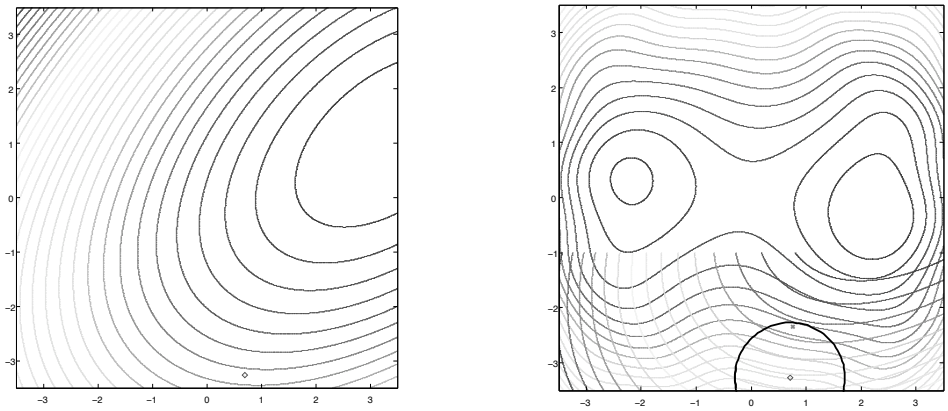


Figure 8.4: $k = 0$. The left image represents the contours of m_0, the right image represents the contours of f and m_0. The open dot is the initial point, the filled dot at $(0.757, -2.336)$ is a point with sufficient decrease. The objective function has a value of 43.742, which is lower than $f(x_0)$ hence we decide to move to this point and call it x_1. We even increase the trust-region radius for the next iteration and we hope that the model will still represent the objective function. This new model is represented in the next figure.

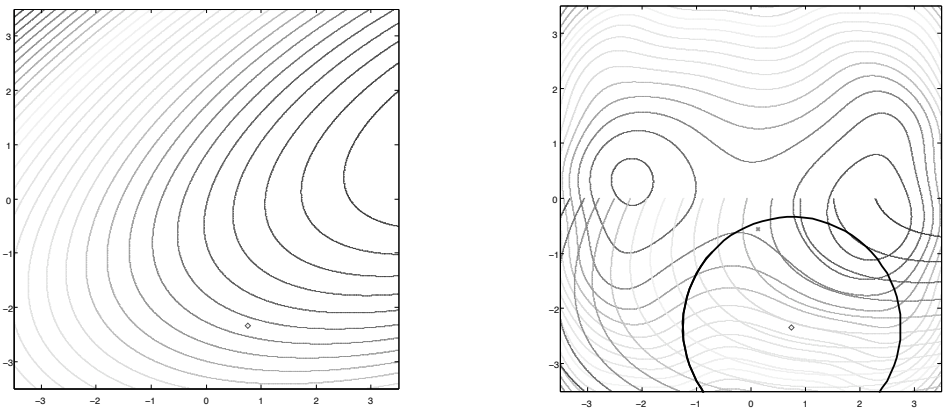


Figure 8.5: $k = 1$. The trial point $(0.141, -0.557)$ is a point of reasonable reduction because $f(x) = (2.306)$. We accept this new point and call it x_2. Again, a new model of the objective function is built using $f(x_2)$, the slope and curvature of f at this point. Again we increase the turst region radius because of good results. This new model is represented in the next figure.

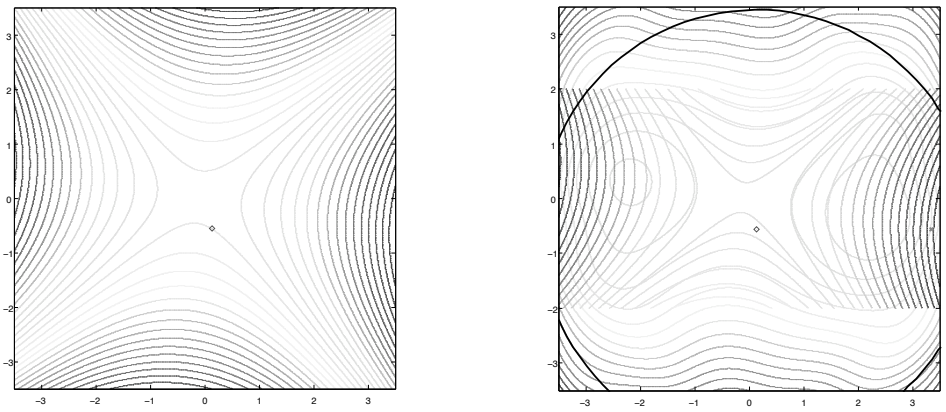


Figure 8.6: $k = 2$. The model no longer represents the objective function well far from x_2. Moving to the indicated trial point may not be a good idea and we stay where we are ($x_3 = x_2$). We reduce the trust-region radius.

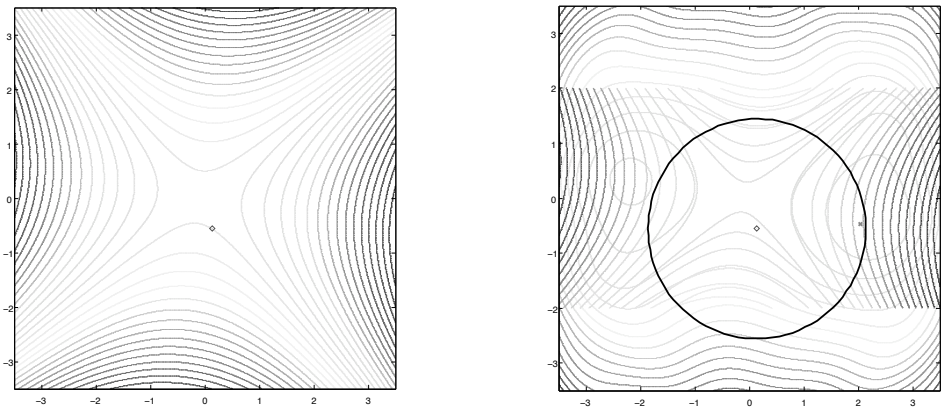


Figure 8.7: $k = 3$. Now we can find a new trial point $(2.0368, -0.4739)$ with sufficient decrease, the value of the objective function is (-29.392). We call the new point x_4, take it as a new initial guess and we keep the current trust-region radius.

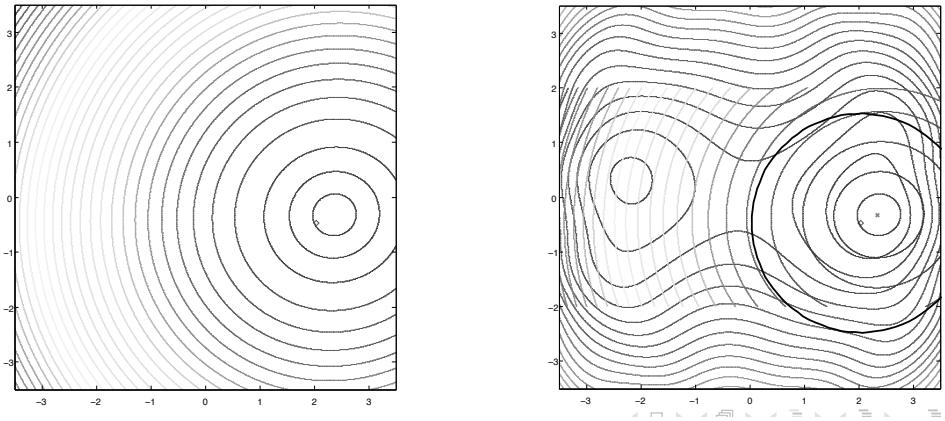


Figure 8.8: $k = 4$. This new model has a minimum at $(2.320, 0.341)$ within the trust-region, which we will choose as our new trial point. This iterate is successful because the model predicts the objective function (-31.131) well at the trial point. For the next iteration, we increase the trust-region radius.

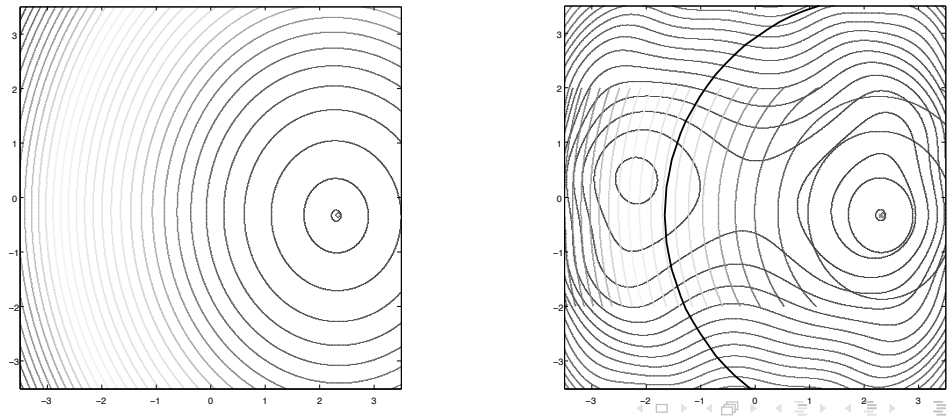


Figure 8.9: $k = 5$. This model minimizer is again successful and is even the last step of the procedure. The algorithm has found the rightmost minimizer of the objective function, that is $x^* = (2.303, -0.341)$ and $f(x^*) = -31.176$.

Chapter 9

Calculating Derivatives

In the previous chapters we saw that we regulary need to calculate ∇f and $\nabla^2 f$. There are several methods for calculating these derivatives:

1. **By hand**

   Expensive and error prone.

2. **Symbolic differentiation**

   Using Mathematica or Maple. The disadvantage is that the result is often a very long code and expensive to evaluate.

3. **Finite differences**

   "Easy and fast, but innacurate"

   This method can always be applied, even if the function to be differentiated is only available as black-box code. To approximate the derivative, we use the fact that for any twice differentiable function

   $$\frac{f(x+tp)-f(x)}{t} = \nabla f(x)^T p + O(t). \tag{9.1}$$

   Thus, we can take the left hand side as an approximation of the directional derivative $\nabla f(x)^T p$. But how should we choose t? If we take t too small, the approximation will suffer from numerical cancellation errors. On the other hand, if we take t too large, the linearization errors will be dominant. A good rule of thumb is to use $t = \sqrt{\varepsilon_{\text{mach}}}$, with $\varepsilon_{\text{mach}}$ the machine precision (or the precision of f, if it is lower than the machine precision).

   The accuracy of this method is $\sqrt{\varepsilon_{\text{mach}}}$, which means in practice that we loose half the valid digits compared to the function evaluation. Second order derivates are even more difficult to accurately calculate.

4. **Algorithmic Differentiation (AD)**

   This is the main topic of this chapter.

9.1 Algorithmic Differentiation (AD)

Algorithmic differentiation uses the fact that each differentiable function $F : \mathbb{R}^n \to \mathbb{R}^{n_F}$ is composed of several *elementary operations*, like multiplication, division, addition, subtraction, sine-functions, exp-functions, etc. If the function is written in a programming language like e.g. C, C++ or FORTRAN, special AD-tools can have access to all these elementary operations. They can process the code in order to generate new code that does not only deliver the function value, but also desired derivative information. Algorithmic differentiation was traditionally called *automatic differentiation*, but as this might lead to confusion with symbolic differentiation, most AD people now prefer the term *algorithmic differentiation*, which fortunately has the same abbreviation. An authoritative textbook on AD is [3].

In order to see how AD works, let us regard a function $F : \mathbb{R}^n \to \mathbb{R}^{n_F}$ that is composed of a sequence of m elementary operations. While the inputs $x_1, \ldots, x_n$ are given before, each elementary operation ϕ_i, $i = 0, \ldots, m-1$ generates another intermediate variable, x_{n+i+1}. Some of these intermediate variables are used as output of the code, which we might call $y_1, \ldots, y_{n_F}$ here. The vector $y \in \mathbb{R}^{n_F}$ can be obtained from the vector of all previous variables $x \in \mathbb{R}^{n+m}$ by the expression $y = Cx$ with a selection matrix $C \in \mathbb{R}^{n_F \times (n+m)}$ that consists only of zeros and ones and has only one one in each row. This way to regard a function evaluation is stated in Algorithm 4 and illustrated in Example 9.1 below.

Algorithm 4 User Function Evaluation via Elementary Operations

Input: $x_1, \ldots, x_n$
Output: $y_1, \ldots, y_{n_F}$

for $i = 0$ to $m - 1$ **do**
  $x_{n+i+1} \leftarrow \phi_i(x_1, \ldots, x_{n+i})$
end for

for $j = 0$ to n_F **do**
  $y_j = \sum_{i=1}^{n+m} C_{ji} x_i$
end for
Remark 1: each ϕ_i depends on only one or two out of $\{x_1, \ldots, x_{n+i}\}$.
Remark 2: the selection of y_j from x_i creates no computational costs.

Example 9.1 (Function Evaluation via Elementary Operations)**:** Let us regard the simple scalar function

$$f(x_1, x_2, x_3) = \sin(x_1 x_2) + \exp(x_1 x_2 x_3)$$

with $n = 3$. We can decompose this function into $m = 5$ elementary operations, namely

$$\begin{aligned} x_4 &= x_1 x_2 \\ x_5 &= \sin(x_4) \\ x_6 &= x_4 x_3 \\ x_7 &= \exp(x_6) \\ x_8 &= x_5 + x_7 \\ y_1 &= x_8 \end{aligned}$$

Thus, if the $n = 3$ inputs x_1, x_2, x_3 are given, the $m = 5$ elementary operations $\phi_0, \ldots, \phi_4$ compute the $m = 5$ intermediate quantities, $x_4, \ldots, x_8$. The last row defines that our desired output is x_8, i.e. the selection matrix C is in this example given by

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The idea of AD is to use the chain rule and differentiate each of the elementary operations ϕ_i separately. There are two modes of AD, on the one hand the "forward" mode of AD, and on the other hand the "backward", "reverse", or "adjoint" mode of AD. In order to present both of them in a consistent form, we first introduce an alternative formulation of the original user function, that uses augmented elementary functions, as follows[1]: we introduce new augmented states

$$\tilde{x}_0 = x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \tilde{x}_1 = \begin{bmatrix} x_1 \\ \vdots \\ x_{n+1} \end{bmatrix}, \quad \ldots, \quad \tilde{x}_m = \begin{bmatrix} x_1 \\ \vdots \\ x_{n+m} \end{bmatrix} \tag{9.2}$$

as well as new augmented elementary functions $\tilde{\phi}_i : \mathbb{R}^{n+i} \to \mathbb{R}^{n+i+1}$, $\tilde{x}_i \mapsto \tilde{x}_{i+1} = \tilde{\phi}_i(\tilde{x}_i)$ with

$$\tilde{\phi}_i(\tilde{x}_i) = \begin{bmatrix} x_1 \\ \vdots \\ x_{n+i} \\ \phi_i(x_1, \ldots, x_{n+i}) \end{bmatrix}, \quad i = 0, \ldots, m-1. \tag{9.3}$$

Thus, the whole evaluation tree of the function can be summarized as a concatenation of these augmented functions followed by a multiplication with the selection matrix C that selects from $\tilde{x}_m$ the final outputs of the computer code.

$$F(x) = C \cdot \tilde{\phi}_{m-1}(\tilde{\phi}_{m-2}(\cdots \tilde{\phi}_1(\tilde{\phi}_0(x)))).$$

The full Jacobian of F, that we denote by $J_F = \frac{\partial F}{\partial x}$, is given by the chain rule as the product of the Jacobians of the augmented elementary functions$\tilde{J}_i = \frac{\partial \tilde{\phi}_i}{\partial \tilde{x}_i}$, as follows:

$$J_F = C \cdot \tilde{J}_{m-1} \cdot \tilde{J}_{m-2} \cdots \tilde{J}_1 \cdot \tilde{J}_0. \tag{9.4}$$

[1]MD thanks Carlo Savorgnan for having outlined to him this way of presenting forward and backward AD

Note that each elementary Jacobian is given as a unit matrix plus one extra row. Also note that the extra row that is here marked with stars $*$ has at maximum two non-zero entries.

$$\tilde{J}_i = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ * & * & * & * & \end{bmatrix}.$$

For the generation of first order derivatives, algorithmic differentiation uses two alternative ways to evaluate the product of these Jacobians, the *forward* and the *backward mode* as described in the next sections.

9.2 The Forward Mode of AD

In forward AD we first define a *seed vector* $p \in \mathbb{R}^n$ and then evaluate the directional derivative $J_F p$ in the following way:

$$J_F p = C \cdot (\tilde{J}_{m-1} \cdot (\tilde{J}_{m-2} \cdots (\tilde{J}_1 \cdot (\tilde{J}_0 p)))). \tag{9.5}$$

In order to write down this long matrix product as an efficient algorithm where the multiplications of all the ones and zeros do not cause computational costs, it is customary in the field of AD to use a notation that uses "dot quantities" $\dot{x}_i$ that we might think of as the velocity with which a certain variable changes, given that the input x changes with speed $\dot{x} = p$. We can interpret them as

$$\dot{x}_i \equiv \frac{dx_i}{dx} p.$$

In the augmented formulation, we can introduce dot quantities $\dot{\tilde{x}}_i$ for the augmented vectors $\tilde{x}_i$, for $i = 0, \ldots, m-1$, and the recursion of these dot quantities is just given by the initialization with the seed vector, $\dot{\tilde{x}}_0 = p$, and then the recursion

$$\dot{\tilde{x}}_{i+1} = \tilde{J}_i(\tilde{x}_i)\dot{\tilde{x}}_i, \quad i = 0, 1, \ldots, m-1.$$

Given the special structure of the Jacobian matrices, most elements of $\dot{\tilde{x}}_i$ are only multiplied by one and nothing needs to be done, apart from the computation of the last component of the new vector $\dot{\tilde{x}}_{i+1}$. This last component is $\dot{x}_{n+i+1}$ Thus, in an efficient implementation, the forward AD algorithm works as the algorithm below. It first sets the seed $\dot{x} = p$ and then proceeds as follows.

In forward AD, the function evaluation and the derivative evaluation can be performed in parallel, which eliminates the need to store any internal information. This is best illustrated using an example.

Algorithm 5 Forward Automatic Differentiation

Input: $\dot{x}_1, \ldots, \dot{x}_n$ and all partial derivatives $\frac{\partial \phi_i}{\partial x_j}$
Output: $\dot{x}_1, \ldots, \dot{x}_{n+m}$

for $i = 0$ to $m-1$ **do**
  $\dot{x}_{n+i+1} \leftarrow \sum_{j=1}^{n+i} \frac{\partial \phi_i}{\partial x_j} \dot{x}_j$
end for

Note: each sum consist of only one or two non-zero entries.

Example 9.2 (Forward Automatic Differentiation)**:** We regard the same example as above, $f(x_1, x_2, x_3) = \sin(x_1 x_2) + \exp(x_1 x_2 x_3)$. First, each intermediate variable has to be computed, and then each line can be differentiated. For given x_1, x_2, x_3 and $\dot{x}_1, \dot{x}_2, \dot{x}_3$, the algorithm proceeds as follows:

$$\begin{aligned} x_4 &= x_1 x_2 & \dot{x}_4 &= \dot{x}_1 x_2 + x_1 \dot{x}_2 \\ x_5 &= \sin(x_4) & \dot{x}_5 &= \cos(x_4)\dot{x}_4 \\ x_6 &= x_4 x_3 & \dot{x}_6 &= \dot{x}_4 x_3 + x_4 \dot{x}_3 \\ x_7 &= \exp(x_6) & \dot{x}_7 &= \exp(x_6)\dot{x}_6 \\ x_8 &= x_5 + x_7 & \dot{x}_8 &= \dot{x}_5 + \dot{x}_7 \end{aligned}$$

The result is $\dot{x}_8 = (\dot{x}_1, \dot{x}_2, \dot{x}_3)\nabla f(x_1, x_2, x_3)$.

It can be proven that the computational cost of Algorithm 5 is smaller than two times the cost of Algorithm 4, or short

$$\text{cost}(J_F p) \leq 2\,\text{cost}(F).$$

If we want to obtain the full Jacobian of F, we need to call Algorithm 5 several times, each time with the seed vector corresponding to one of the n unit vectors in $\mathbb{R}^n$, i.e.,

$$\dot{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}. \tag{9.6}$$

Thus, we have for the computation of the full jacobian

$$\text{cost}(J_F) \leq 2\,n\,\text{cost}(F).$$

AD in forward mode is slightly more expensive than numerical finite differences, but it is exact up to machine precision.

The "Imaginary trick" in MATLAB

An easy way to obtain high precision derivatives in MATLAB is closely related to AD in forward mode. It is based on the following observation: if $F : \mathbb{R}^n \to \mathbb{R}^{n_F}$ is analytic and can be extended to complex numbers as inputs and outputs, then for any $t > 0$ holds

$$J_F(x)p = \frac{\text{imag}(F(x + itp))}{t} + O(t^2). \tag{9.7}$$

Proof. We define a function g in a complex scalar variable $z \in \mathbb{C}$ as $g(z) = F(x + zp)$ and then look at its Taylor expansion:

$$\begin{aligned} g(z) &= g(0) + g'(0)z + \frac{1}{2}g''(0)z^2 + O(z^3) \\ g(it) &= g(0) + g'(0)it + \frac{1}{2}g''(0)i^2t^2 + O(t^3) \\ &= g(0) - \frac{1}{2}g''(0)t^2 + g'(0)it + O(t^3) \\ \text{imag}(g(it)) &= g'(0)t + O(t^3) \end{aligned}$$

□

In contrast to finite differences, there is no subtraction in the numerator, so there is no danger of numerical cancellation errors, and t can be chosen extremely small, e.g. $t = 10^{-100}$, which means that we can compute the derivative up to machine precision. This "imaginary trick" can most easily be used in a programming language like MATLAB that does not declare the type of variables beforehand, so that real-valued variables can automatically be overloaded with complex-valued variables. This allows us to obtain high-precision derivatives of a given black-box MATLAB code. We only need to be sure that the code is analytic (which most codes are) and that matrix or vector transposes are not expressed by a prime `'` (which conjugates a complex number), but by `transp()`.

9.3 The Backward Mode of AD

In backward AD we evaluate the product in Eq. (9.4) in the reverse order compared with forward AD. Backward AD does not evaluate forward directional derivatives. Instead, it evaluates *adjoint directional derivatives*: when we define a *seed vector* $\lambda \in \mathbb{R}^{n_F}$ then backward AD is able to evaluate the product $\lambda^T J_F$. It does so in the following way:

$$\lambda^T J_F = ((((\lambda^T C) \cdot \tilde{J}_{m-1}) \cdot \tilde{J}_{m-2}) \cdots \tilde{J}_1) \cdot \tilde{J}_0. \tag{9.8}$$

When writing this matrix product as an algorithm, we use "bar quantities" instead of the "dot quantities" that we used in the forward mode. These quantities can be interpreted as derivatives of the final output with respect to the respective intermediate quantity. We can interpret

$$\bar{x}_i \equiv \lambda^T \frac{dF}{dx_i}.$$

Each intermediate variable has a bar variable and at the start, we initialize all bar variables with the value that we obtain from $C^T\lambda$. Note that most of these seeds will usually be zero, depending on the output selection matrix C. Then, the backward AD algorithm modifies all bar variables. Backward AD gets most transparent in the augmented formulation, where we have bar quantities $\bar{\tilde{x}}_i$ for the augmented states $\tilde{x}_i$. We can transpose the above Equation (9.8) in order to obtain

$$J_F^T\lambda = \tilde{J}_0^T \cdot (\tilde{J}_1^T \cdots \underbrace{\tilde{J}_{m-1}^T \underbrace{(C^T\lambda)}_{=\bar{\tilde{x}}_m}}_{=\bar{\tilde{x}}_{m-1}}).$$

In this formulation, the initialization of the backward seed is nothing else than setting $\bar{\tilde{x}}_m = C^T\lambda$ and then going in reverse order through the recursion

$$\bar{\tilde{x}}_i = \tilde{J}_i(\tilde{x}_i)^T\bar{\tilde{x}}_{i+1}, \quad i = m-1, m-2, \ldots, 0.$$

Again, the multiplication with ones does not cause any computational cost, but an interesting feature of the reverse mode is that some of the bar quantities can get several times modified in very different stages of the algorithm. Note that the multiplication $\tilde{J}_i^T\bar{\tilde{x}}_{i+1}$ with the transposed Jacobian

$$\tilde{J}_i^T = \begin{bmatrix} 1 & & & * \\ & 1 & & * \\ & & \ddots & * \\ & & & 1 \quad * \end{bmatrix}.$$

modifies at maximum two elements of the vector $\bar{\tilde{x}}_{i+1}$ by adding to them the partial derivative of the elementary operation multiplied with $\bar{x}_{n+i+1}$. In an efficient implementation, the backward AD algorithm looks as follows.

Algorithm 6 Reverse Automatic Differentiation

Input: seed vector $\bar{x}_1, \ldots, \bar{x}_{n+m}$ and all partial derivatives $\frac{\partial \phi_i}{\partial x_j}$
Output: $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$

for $i = m-1$ down to 0 **do**
  for all $j = 1, \ldots, n+i$ **do**
    $\bar{x}_j \leftarrow \bar{x}_j + \bar{x}_{n+i+1}\frac{\partial \phi_i}{\partial x_j}$
  end for
end for

Note: each inner loop will only update one or two bar quantities.

Example 9.3 (Reverse Automatic Differentiation)**:** We regard the same example as before, and want to compute the gradient $\nabla f(x) = (\bar{x}_1, \bar{x}_2, \bar{x}_3)^T$ given (x_1, x_2, x_3). We set $\lambda = 1$. Because the selection matrix C selects only the last intermediate variable as output, i.e. $C = (0, \cdots 0, 1)$, we initialize the seed vector with zeros apart from the last component, which is one. In the reverse mode, the algorithm first has to evaluate the function with all intermediate quantities, and only

then it can compute the bar quantities, which it does in reverse order. At the end it obtains, among other, the desired quantitities $(\bar{x}_1, \bar{x}_2, \bar{x}_3)$. The full algorithm is the following.

// *** forward evaluation of the function ***
$x_4 = x_1 x_2$
$x_5 = \sin(x_4)$
$x_6 = x_4 x_3$
$x_7 = \exp(x_6)$
$x_8 = x_5 + x_7$

// *** initialization of the seed vector ***
$\bar{x}_i = 0, \quad i = 1, \ldots, 7$
$\bar{x}_8 = 1$

// *** backwards sweep ***
// * differentiation of $x_8 = x_5 + x_7$
$\bar{x}_5 = \bar{x}_5 + 1\,\bar{x}_8$
$\bar{x}_7 = \bar{x}_7 + 1\,\bar{x}_8$
// * differentiation of $x_7 = \exp(x_6)$
$\bar{x}_6 = \bar{x}_6 + \exp(x_6)\bar{x}_7$
// * differentiation of $x_6 = x_4 x_3$
$\bar{x}_4 = \bar{x}_4 + x_3\bar{x}_6$
$\bar{x}_3 = \bar{x}_3 + x_4\bar{x}_6$
// * differentiation of $x_5 = \sin(x_4)$
$\bar{x}_4 = \bar{x}_4 + \cos(x_4)\bar{x}_5$
// differentiation of $x_4 = x_1 x_2$
$\bar{x}_1 = \bar{x}_1 + x_2\bar{x}_4$
$\bar{x}_2 = \bar{x}_2 + x_1\bar{x}_4$

The desired output of the algorithm is $(\bar{x}_1, \bar{x}_2, \bar{x}_3)$, equal to the three components of the gradient $\nabla f(x)$. Note that all three are returned in *only one* reverse sweep.

It can be shown that the cost of Algorithm 6 is less than 3 times the cost of Algorithm 4, i.e.,

$$\text{cost}(\lambda^T J_F) \leq 3\,\text{cost}(F).$$

If we want to obtain the full Jacobian of F, we need to call Algorithm 6 several times with the n_F seed vectors corresponding to the unit vectors in $\mathbb{R}^{n_F}$, i.e. we have

$$\text{cost}(J_F) \leq 3\,n_F\,\text{cost}(F).$$

This is a remarkable fact: it means that the backward mode of AD can compute the full Jacobian at a cost that is independent of the state dimension n. This is particularly advantageous if $n_F \ll n$,

e.g. if we compute the gradient of a scalar function like the objective or the Lagrangian. The reverse mode can be much faster than what we can obtain by finite differences, where we always need $(n+1)$ function evaluations. To give an example, if we want to compute the gradient of a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ with n =1 000 000 and each call of the function needs one second of CPU time, then the finite difference approximation of the gradient would take 1 000 001 seconds, while the computation of the same quantity with the backward mode of AD needs only 4 seconds (1 call of the function plus one backward sweep). Thus, besides being more accurate, backward AD can also be much faster than finite differences.

The only disadvantage of the backward mode of AD is that we have to store all intermediate variables and partial derivatives, in contrast to finite differences or forward AD. A partial remedy to this problem exist in form of *checkpointing* that trades-off computational speed and memory requirements. Instead of all intermediate variables, it only stores some "checkpoints" during the forward evaluation. During the backward sweep, starting at these checkpoints, it re-evaluates parts of the function to obtain those intermediate variables that have not been stored. The optimal number and location of checkpoints is a science of itself. Generally speaking, checkpointing reduces the memory requirements, but comes at the expense of runtime.

From a user perspective, the details of implementation are not too relevant, but it is most important to just know that the reverse mode of AD exists and that it allows in many cases a much more efficient derivative generation than any other technique.

Efficient Computation of the Hessian

A particularly important quantity in Newton-type optimization methods is the Hessian of the Lagrangian. It is the second derivative of the scalar function $\mathcal{L}(x, \lambda, \mu)$ with respect to x. As the multipliers are fixed for the purpose of differentiation, we can for notational simplicity just regard a function $f : \mathbb{R}^n \to \mathbb{R}$ of which we want to compute the Hessian $\nabla^2 f(x)$. With finite differences we would at least need $n(n+1)/2$ function evaluations in order to compute the Hessian, and due to round-off and truncation errors, the accuracy of a finite difference Hessian would be much lower than the accuracy of the function f.

In contrast to this, algorithmic differentiation can without problems be applied recursively, yielding a code that computes the Hessian matrix at the same precision as the function f itself, i.e. typically at machine precision. Moreover, if we use the reverse mode of AD at least once, e.g. by first generating an efficient code for $\nabla f(x)$ (using backward AD) and then using forward AD to obtain the Jacobian of it, we can reduce the CPU time considerably compared to finite differences. Using the above procedure, we would obtain the Hessian $\nabla^2 f$ at a cost of $2\,n$ times the cost of a gradient ∇f, which is about four times the cost of evaluating f alone. This means that we have the following runtime bound:

$$\text{cost}(\nabla^2 f) \leq 8\, n\, \text{cost}(f).$$

A compromise between accuracy and ease of implementation that is equally fast in terms of CPU time is to use backward AD only for computing the first order derivative $\nabla f(x)$, and then to use finite differences for the differentiation of $\nabla f(x)$.

9.4 Algorithmic Differentiation Software

Most algorithmic differentiation tools implement both forward and backward AD, and most are specific to one particular programming language. They come in two different variants: either they use *operator overloading* or *source-code transformation*.

The first class does not modify the code but changes the type of the variables and overloads the involved elementary operations. For the forward mode, each variable just gets an additional dot-quantity, i.e. the new variables are the pairs $(x_i, \dot{x}_i)$, and elementary operations just operate on these pairs, like e.g.

$$(x, \dot{x}) \cdot (y, \dot{y}) = (xy, x\dot{y} + y\dot{x}).$$

An interesting remark is that operator overloading is also at the basis of the imaginary trick in MATLAB were we use the overloading of real numbers by complex numbers and used the small imaginary part as dot quantity and exploited the fact that the extremely small higher order terms disappear by numerical cancellation.

A prominent and widely used AD tool for generic user supplied C++ code that uses operator overloading is ADOL-C. Though it is not the most efficient AD tool in terms of CPU time it is well documented and stable. Another popular tool in this class is CppAD.

The other class of AD tools is based on source-code transformation. They work like a text-processing tool that gets as input the user supplied source code and produces as output a new and very differently looking source code that implements the derivative generation. Often, these codes can be made extremely fast. Tools that implement source code transformations are ADIC for ANSI C, and ADIFOR and TAPENADE for FORTRAN codes.

In the context of simulation of ordinary differential equations (ODE), there exist good numerical integrators with forward and backward differentiation capabilities that are more efficient and reliable than a naive procedure that would consist of taking an integrator and processing it with an AD tool. Examples for integrators that use the principle of forward and backward AD are the code DAESOL-II or the open-source codes from the ACADO Integrators Collection or from the SUNDIALS Suite. Another interesting AD tool, CasADi, is an optimization modelling language that implements all variants of AD and provides several interfaces to ODE solvers with forward and derivative computations, as well as to optimization codes. It can conveniently be used from a Python front end.

Part III

Constrained Optimization Algorithms

Chapter 10

Optimality Conditions for Constrained Optimization

From now on, regard the general constrained minimization problem in standard form

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) && && \text{(10.1a)} \\ \text{subject to} \quad & g(x) & = & \ 0, && \text{(10.1b)} \\ & h(x) & \geq & \ 0. && \text{(10.1c)} \end{aligned}$$

in which $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^q$ are smooth. Recall that the feasible set for this problem is defined as $\Omega = \{x \in \mathbb{R}^n | g(x) = 0,\ h(x) \geq 0\}$.

Definition 10.1 (Tangent)
$p \in \mathbb{R}^n$ is called a "tangent" to Ω at $x^* \in \Omega$ if there exists a smooth curve $\bar{x}(t) : [0, \epsilon) \to \mathbb{R}^n$ with $\bar{x}(0) = x^*, \bar{x}(t) \in \Omega\ \forall t \in [0, \epsilon)$ and $\frac{d\bar{x}}{dt}(0) = p$.

Definition 10.2 (Tangent Cone)
the "tangent cone" $T_\Omega(x^*)$ of Ω at x^* is the set of all tangent vectors at x^*.

Example 10.1 (Tangent Cone)**:** Regard $\Omega = \left\{x \in \mathbb{R}^2 | h(x) \geq 0\right\}$ with

$$h(x) = \begin{bmatrix} (x_1-1)^2 + x_2^2 - 1 \\ -(x_2-2)^2 - x_1^2 + 4 \end{bmatrix} \tag{10.2}$$

$$x* = \begin{bmatrix} 0 \\ 4 \end{bmatrix} : T_\Omega(x^*) = \left\{ p | p^T \begin{bmatrix} 0 \\ -1 \end{bmatrix} \geq 0 \right\} = \mathbb{R} \times \mathbb{R}_{--} \tag{10.3}$$

$$x* = \begin{bmatrix} 0 \\ 0 \end{bmatrix} : T_\Omega(x^*) = \left\{ p | p^T \begin{bmatrix} -1 \\ 0 \end{bmatrix} \geq 0 \;\&\; p^T \begin{bmatrix} 0 \\ 1 \end{bmatrix} \geq 0 \right\} = \mathbb{R}_{--} \times \mathbb{R}_{++} \tag{10.4}$$

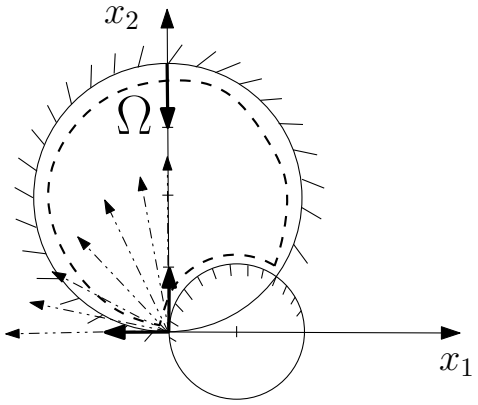


Figure 10.1: Visualization of Example 10.1.

In this example, we can generate the tangent cone by hand, making a sketch and afterwards defining the sets accordingly using suitable inequalities. But we will soon see a much more powerful way to generate the tangent cone directly by a linearization of the nonlinear inequalities. This is however, only possible under some condition, which we will call "constraint qualification". Before, let us see for what aim serves the tangent cone.

10.1 Karush-Kuhn-Tucker (KKT) Necessary Optimality Conditions

Theorem 10.1 (First Order Necessary Conditions, variant 0)**:** *If x^* is a local minimum of the NLP (10.1) then*

1. $x^* \in \Omega$
2. *for all tangents $p \in T_\Omega(x^*)$ holds:* $\nabla f(x^*)^T p \geq 0$

Proof by contradiction. If $\exists p \in T_\Omega(x^*)$ with $\nabla f(x^*)^T p < 0$ there would exist a feasible curve $\bar{x}(t)$ with $\frac{df(\bar{x}(t))}{dt}\big|_{t=0} = \nabla f(x^*)^T p < 0$. □

10.2 Active Constraints and Constraint Qualification

How can we characterize $T_\Omega(x^*)$?

Definition 10.3 (Active/Inactive Constraint)
An inequality constraint $h_i(x) \geq 0$ is called "active" at $x^* \in \Omega$ iff $h_i(x^*) = 0$ and otherwise "inactive".

Definition 10.4 (Active Set)
The index set $\mathcal{A}(x^*) \subset \{1, \ldots, q\}$ of active constraints is called the "active set".

Remark. Inactive constraints do not influence $T_\Omega(x^*)$.

Definition 10.5 (LICQ)
The "linear independence constraint qualification" (LICQ) holds at $x^* \in \Omega$ iff all vectors $\nabla g_i(x^*)$ for $i \in \{1, \ldots, m\}$ & $\nabla h_i(x^*)$ for $i \in \mathcal{A}(x^*)$ are linearly independent.

Remark. this is a technical condition, and is usually satisfied.

Definition 10.6 (Linearized Feasible Cone)
$\mathcal{F}(x^*) = \{p | \nabla g_i(x^*)^T p = 0,\ i = 1, \ldots, m\ \&\ \nabla h_i(x^*)^T p \geq 0,\ i \in \mathcal{A}(x^*)\}$ is called the "linearized feasible cone" at $x^* \in \Omega$.

Example 10.2 (Linearized Feasible Cone)**:**

$$h(x) = \begin{bmatrix} (x_1-1)^2 + x_2^2 - 1 \\ -(x_2-2)^2 - x_1^2 + 4 \end{bmatrix} \tag{10.5}$$

$$x^* = \begin{bmatrix} 0 \\ 4 \end{bmatrix}, \quad \mathcal{A}(x^*) = \{2\} \tag{10.6}$$

$$\nabla h_2(x) = \begin{bmatrix} -2x_1 \\ -2(x_2-2) \end{bmatrix} \tag{10.7}$$

$$= \begin{bmatrix} 0 \\ -4 \end{bmatrix} \tag{10.8}$$

$$\mathcal{F}(x^*) = \left\{ p | \begin{bmatrix} 0 \\ -4 \end{bmatrix}^T p \geq 0 \right\} \tag{10.9}$$

Theorem 10.2: *At any $x^* \in \Omega$ holds*

1. $T_\Omega(x^*) \subset \mathcal{F}(x^*)$
2. *If LICQ holds at* x^* *then* $T_\Omega(x^*) = \mathcal{F}(x^*)$.

Sketch of proof:

1. Sketch:

$$p \in T_\Omega \Rightarrow \exists \bar{x}(t) \text{ with } p = \frac{d\bar{x}}{dt}\Big|_{t=0} \;\&\; \bar{x}(0) = x^* \;\&\; \bar{x}(t) \in \Omega \tag{10.10}$$

$$\begin{aligned}
\Rightarrow \quad & g(\bar{x}(t)) = 0 \quad \text{and} && (10.11)\\
& h(\bar{x}(t)) \geq 0 \quad \forall t \in [0, \epsilon) && (10.12)\\
\Rightarrow \quad & \frac{dg_i(\bar{x}(t))}{dt}\Big|_{t=0} = \nabla g_i(x^*)^T p = 0,\ i = 1, \ldots, m, \quad \text{and} && (10.13)\\
& \frac{dh_i(\bar{x}(t))}{dt}\Big|_{t=0} = \lim_{t \to 0+} \frac{h_i(\bar{x}(t)) - h_i(x^*)}{t} \geq 0 \quad \text{for } i \in \mathcal{A}(x^*) && (10.14)\\
& \Leftrightarrow \frac{dh_i(\bar{x}(t))}{dt}\Big|_{t=0} = \nabla h_i(x^*)^T p \geq 0 && (10.15)\\
\Rightarrow \quad & p \in \mathcal{F}(x^*) && (10.16)
\end{aligned}$$

2. For the full proof see [Noc2006]. The idea is to use the implicit function theorem to construct a curve $\bar{x}(t)$ which has a given vector $p \in \mathcal{F}(x^*)$ as tangent.

Theorem 10.3 (FONC, Variant 1)**:** *If LICQ holds at* x^* *and* x^* *is a local minimizer for the NLP (10.1) then*

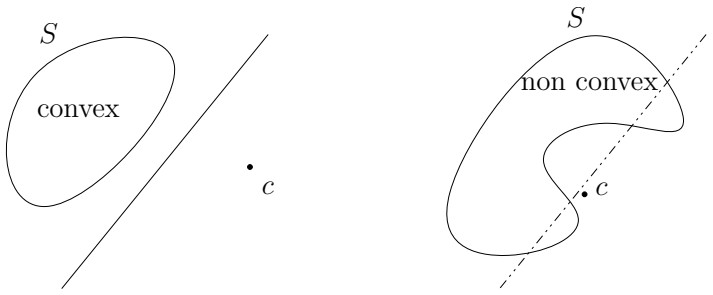
1. $x^* \in \Omega$
2. $\forall p \in \mathcal{F}(x^*) : \nabla f(x^*)^T p \geq 0$.

How can we simplify the second condition? Here helps the following lemma. To interpret it, remember that $\mathcal{F}(x^*) = \{p | Gp = 0,\ Hp \geq 0\}$ *with* $G = \frac{dg}{dx}(x^*)$, $H = \begin{bmatrix} \nabla h_i(x^*)^T \\ \vdots \end{bmatrix}$ *for* $i \in \mathcal{A}(x^*)$.

Lemma 10.4 (Farkas' Lemma)**:** *For any matrices* $G \in \mathbb{R}^{m \times n}$, $H \in \mathbb{R}^{q \times n}$ *and vector* $c \in \mathbb{R}^n$ *holds*

$$\begin{aligned}
\textit{either} \quad & \exists \lambda \in \mathbb{R}^m, \mu \in \mathbb{R}^q \text{ with } \mu \geq 0 \;\&\; c = G^T\lambda + H^T\mu && (10.17)\\
\textit{or} \quad & \exists p \in \mathbb{R}^n \text{ with } Gp = 0 \;\&\; Hp \geq 0 \;\&\; c^T p < 0 && (10.18)
\end{aligned}$$

but never both ("theorem of alternatives").



(a) convex (b) non convex

Figure 10.2: Visualization of the separating hyperplane Theorem, used in the proof of Lemma 10.4. For the non convex case, no hyperplane can be found.

Proof. In the proof we use the "separating hyperplane theorem" with respect to the point $c \in \mathbb{R}^n$ and the set $S = \{G^T\lambda + H^T\mu | \lambda \in \mathbb{R}^m,\ \mu \in \mathbb{R}^q,\ \mu \geq 0\}$. S is a convex cone. The separating hyperplane theorem states that two convex sets – in our case the set S and the point c – can always be separated by a hyperplane. In our case, the hyperplane touches the set S at the origin, and is described by a normal vector p. Separation of S and c means that for all $y \in S$ holds that $y^T p \geq 0$ and on the other hand, $c^T p < 0$.

$$\begin{aligned}
\text{Either} \quad & c \in S \Leftrightarrow (10.17) && (10.19)\\
\text{or} \quad & c \notin S && (10.20)\\
& \Leftrightarrow \exists p \in \mathbb{R}^n : \forall y \in S : p^T y \geq 0 \ \&\ p^T c < 0 && (10.21)\\
& \Leftrightarrow \exists p \in \mathbb{R}^n : \forall \lambda, \mu \text{ with } \mu \geq 0 : p^T(G^T\lambda + H^T\mu) \geq 0 \ \&\ p^T c < 0 && (10.22)\\
& \Leftrightarrow \exists p \in \mathbb{R}^n : Gp = 0 \ \&\ Hp \geq 0 \ \&\ p^T c < 0 \Leftrightarrow (10.18) && (10.23)
\end{aligned}$$

The last line follows because

$$\begin{aligned}
& \forall \lambda, \mu \text{ with } \mu \geq 0 : p^T(G^T\lambda + H^T\mu) \geq 0 && (10.24)\\
& \Leftrightarrow \forall \lambda : \lambda^T Gp \geq 0 \text{ and } \forall \mu \geq 0 : \mu^T Hp \geq 0 && (10.25)\\
& \Leftrightarrow Gp = 0 \ \&\ Hp \geq 0. && (10.26)
\end{aligned}$$

□

From Farkas' lemma follows the desired simplification of the previous theorem:

Theorem 10.5 (FONC, Variant 2: KKT Conditions)**:** *If x^* is a local minimizer of the NLP (10.1) and LICQ holds at x^* then there exists a $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^q$ with*

$$\begin{aligned}
\nabla f(x^*) - \nabla g(x^*)\lambda^* - \nabla h(x^*)\mu^* &= 0 && (10.27a)\\
g(x^*) &= 0 && (10.27b)\\
h(x^*) &\geq 0 && (10.27c)\\
\mu^* &\geq 0 && (10.27d)\\
\mu_i^* h_i(x^*) &= 0, \quad i = 1, \ldots, q. && (10.27e)
\end{aligned}$$

Note: The KKT conditions are the First order necessary conditions for optimality (FONC) for constrained optimization, and are thus the equivalent to $\nabla f(x^*) = 0$ in unconstrained optimization.

Proof. We know already that (10.27b), (10.27c) $\Leftrightarrow x^* \in \Omega$. We have to show that (10.27a), (10.27d), (10.27e) $\Leftrightarrow \forall p \in \mathcal{F}(x^*) : p^T \nabla f(x^*) \geq 0$. Using Farkas' lemma we have

$$\begin{aligned} \forall p \in \mathcal{F}(x^*) : p^T \nabla f(x^*) \geq 0 \quad &\Leftrightarrow \quad \text{It is not true that } \exists p \in \mathcal{F}(x^*) : p^T \nabla f(x^*) < 0 \\ &\Leftrightarrow \quad \exists \lambda^*, \mu_i^* \geq 0 : \ \nabla f(x^*) = \sum \nabla g_i(x^*) \lambda_i^* + \sum_{i \in \mathcal{A}(x^*)} \nabla h_i(x^*) \mu_i^* \end{aligned}$$

Now we set all components of μ that are not element of $\mathcal{A}(x^*)$ to zero, i.e. $\mu_i = 0$ if $h_i(x^*) > 0$, and conditions (10.27d) and (10.27e) are trivially satisfied, as well as (10.27a) due to $\sum_{i \in \mathcal{A}(x^*)} \nabla h_i(x^*) \mu_i^* = \sum_{i=\{1,\ldots,q\}} \nabla h_i(x^*) \mu_i$ if $\mu_i^* = 0$ for $i \notin \mathcal{A}(x^*)$. □

Though it is not necessary for the proof of the *necessity* of the optimality conditions of the above theorem (variant 2), we point out that the theorem is 100 % equivalent to variant 1, but has the computational advantage that its conditions can be checked easily: if someone gives you a triple (x^*, λ^*, μ^*) you can check if it is a KKT point or not.

Note: Using the definition of the Lagrangian, we have (10.27a) $\Leftrightarrow \nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = 0$. In absence of inequalities, the KKT conditions simplify to $\nabla_x \mathcal{L}(x, \lambda) = 0$, $g(x) = 0$, a formulation that is due to Lagrange and was much earlier known than the KKT conditions.

Example 10.3 (KKT Condition)**:**

$$\underset{x \in \mathbb{R}^2}{\text{minimize}} \quad \begin{bmatrix} 0 \\ -1 \end{bmatrix}^T x \tag{10.28}$$

$$\text{subject to} \quad \begin{bmatrix} x_1^2 + x_2^2 - 1 \\ -(x_2 - 2)^2 - x_1^2 + 4 \end{bmatrix} \geq 0 \tag{10.29}$$

Does the local minimizer $x^* = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$ satisfy the KKT conditions?

First:

$$\mathcal{A}(x^*) = \{2\} \tag{10.30}$$

$$\nabla f(x^*) = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \tag{10.31}$$

$$\nabla h_2(x^*) = \begin{bmatrix} 0 \\ -4 \end{bmatrix} \tag{10.32}$$

Then we write down the KKT conditions, which are for the specific dimensions of this example equivalent to the right hand side terms:

$$\begin{aligned}
&(10.27a) \quad \Leftrightarrow \quad \nabla f(x^*) - \nabla h_1(x^*)\mu_1^* - \nabla h_2(x^*)\mu_2^* = 0 && (10.33)\\
&(10.27b) \quad\quad - && (10.34)\\
&(10.27c) \quad \Leftrightarrow \quad h_1(x^*) \geq 0 \;\&\; h_2(x^*) \geq 0 && (10.35)\\
&(10.27d) \quad \Leftrightarrow \quad \mu_1 \geq 0 \;\&\; \mu_2 \geq 0 && (10.36)\\
&(10.27e) \quad \Leftrightarrow \quad \mu_1 h_1(x^*) = 0 \;\&\; \mu_2 h_2(x^*) = 0 && (10.37)
\end{aligned}$$

Finally we check that indeed, all five conditions are satisfied, if we choose μ_1^* and μ_2^* suitably

$$\begin{aligned}
&(10.27a) \quad \Leftarrow \quad \begin{bmatrix} 0 \\ -1 \end{bmatrix} - \begin{bmatrix} * \\ * \end{bmatrix} \mu_1 - \begin{bmatrix} 0 \\ -4 \end{bmatrix} \mu_2 = 0 \text{ (μ_1 is inactive, use } \mu_1^* = 0,\ \mu_2^* = \tfrac{1}{4}) && (10.38)\\
&(10.27b) \quad\quad - && (10.39)\\
&(10.27c) \quad \Leftarrow \quad h_1(x^*) > 0 \;\&\; h_2(x^*) = 0 && (10.40)\\
&(10.27d) \quad \Leftarrow \quad \mu_1 = 0 \;\&\; \mu_2 = \frac{1}{4} \geq 0 && (10.41)\\
&(10.27e) \quad \Leftarrow \quad \mu_1 h_1(x^*) = 0 h_1(x^*) = 0 \;\&\; \mu_2 h_2(x^*) = \mu_2 0 = 0 && (10.42)
\end{aligned}$$

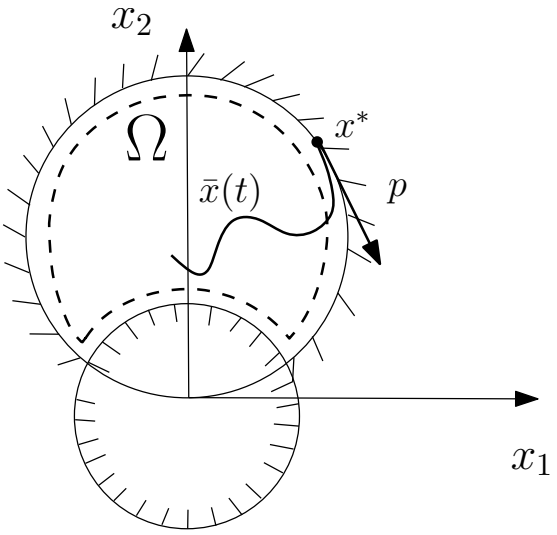


Figure 10.3: Visualization of Example 10.3.

10.3 Convex Problems

Theorem 10.6: *Regard a convex NLP and a point x^* at which LICQ holds. Then:*

$$x^* \text{ is global minimizer} \iff \exists \lambda, \mu \text{ so that KKT condition hold.}$$

Recall that the NLP

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) && && (10.43)\\ \text{subject to} \quad & g(x) &=& \ 0, && (10.44)\\ & -h(x) &\leq& \ 0. && (10.45) \end{aligned}$$

is convex if f and all $-h_i$ are convex and g is affine, i.e., $g(x) = Gx + a$.

Sketch of proof: We only need the "$\Leftarrow$"-direction.

- Assume (x^*, λ^*, μ^*) satisfies the KKT conditions
- $\mathcal{L}(x, \lambda, \mu) = f(x) - \sum g_i(x)\lambda_i - \sum h_i(x)\mu_i$
- $\mathcal{L}$ is a convex function of x, and for fixed μ^*, λ^* its gradient is zero, $\nabla\mathcal{L}(x^*, \lambda^*, \mu^*) = 0$. Therefore, x^* is a global minimizer of the unconstrained minimization problem $\min_x \mathcal{L}(x, \lambda^*, \mu^*) = d^*$
- We know that
  $d^* \leq p^* = \min f(x)$ st $g(x) = 0$, $h(x) \geq 0$.
  $d^* = \mathcal{L}(x^*, \lambda^*, \mu^*) = f(x^*) - \underbrace{\sum g_i(x^*)\lambda_i^*}_{=0} - \underbrace{\sum h_i(x^*)\mu_i^*}_{=0} = f(x^*)$ and
  x^* is feasible: i.e. $p^* = d^*$ and x^* is global minimizer. □

10.4 Complementarity

The last KKT condition (10.27e) is called the *complementarity* condition. The situation for $h_i(x)$ and μ_i that satisfy the three conditions $h_i \geq 0$, $\mu_i \geq 0$ and $h_i\mu_i = 0$ is visualized in Figure 10.4.

Definition 10.7
Regard a KKT point (x^*, λ, μ). For $i \in \mathcal{A}(x^*)$ we say h_i is *weakly active* if $\mu_i = 0$, otherwise, if $\mu_i > 0$, we call it *strictly active*. We say that *strict complementarity* holds at this KKT point iff all active constraints are strictly active. We define the set of weakly active constraints to be $\mathcal{A}_0(x^*, \mu)$ and the set of strictly active constraints $\mathcal{A}_+(x^*, \mu)$. The sets are disjoint and $\mathcal{A}(x^*) = \mathcal{A}_0(x^*, \mu) \cup \mathcal{A}_+(x^*, \mu)$.

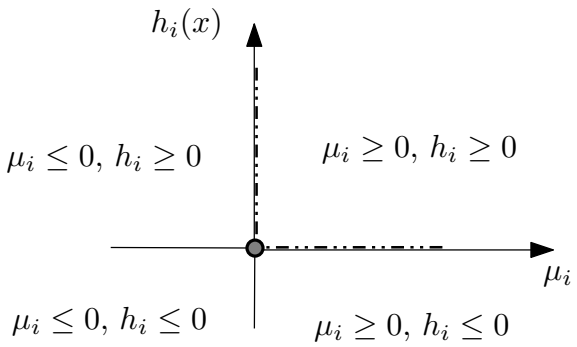


Figure 10.4: The complementarity condition. The origin, $h_i = 0$ and $\mu_i = 0$ makes the complementarity condition non-smooth. Note that strict complementarity makes many theorems easier because it avoids the origin.

10.5 Second Order Conditions

Definition 10.8
Regard the KKT point (x^*, λ, μ). The critical cone $C(x^*, \mu)$ is the following set:

$$C(x^*, \mu) = \{\ p | \nabla g_i(x^*)^T p = 0,\ \nabla h_i(x^*)^T p = 0 \text{ if } i \in \mathcal{A}_+(x^*, \mu),\ \nabla h_i(x^*)^T p \geq 0 \text{ if } i \in \mathcal{A}_0(x^*, \mu)\} \tag{10.46}$$

Note $C(x^*, \mu) \subset \mathcal{F}(x^*)$. In case that LICQ holds, even $C(x^*, \mu) \subset T_\Omega(x^*)$. Thus, the critical cone is a subset of all feasible directions. In fact: it contains all feasible directions which are from first order information neither uphill or downhill directions, as the following theorem shows.

Theorem 10.7 (Criticality of Critical Cone)**:** *Regard the KKT point (x^*, λ, μ) with LICQ, then $\forall p \in T_\Omega(x^*)$ holds*

$$p \in C(x^*, \mu) \ \Leftrightarrow\ \nabla f(x^*)^T p = 0. \tag{10.47}$$

Proof. Use $\nabla_x \mathcal{L}(x^*, \lambda, \mu) = 0$ to get for any $p \in C(x^*, \mu)$:

$$\nabla f(x^*)^T p = \lambda^T \underbrace{\nabla g^T p}_{=0} + \sum_{i,\ \mu_i > 0} \mu_i \underbrace{\nabla h_i(x^*)^T p}_{=0} + \sum_{i,\ \mu_i = 0} \mu_i \nabla h_i(x^*)^T p = 0 \tag{10.48}$$

Conversely, if $p \in T_\Omega(x^*)$ then all terms on the right hand side must be non-negative, so that $\nabla f(x^*)^T p = 0$ implies in particular $\sum_{i,\ \mu_i > 0} \mu_i \nabla h_i(x^*)^T p = 0$ which implies $\nabla h_i(x^*)^T p = 0$ for all $i \in \mathcal{A}_+(x^*, \mu)$, i.e. $p \in C(x^*, \mu)$.

□

Example 10.4:

$$\min x_2 \;\text{ s.t. }\; 1 - x_1^2 - x_2^2 \geq 0 \tag{10.49}$$

$$x^* = \begin{pmatrix} 0 \\ -1 \end{pmatrix} \tag{10.50}$$

$$\nabla h(x) = \begin{pmatrix} -2x_1 \\ -2x_2 \end{pmatrix} \tag{10.51}$$

$$\nabla f(x) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{10.52}$$

$\mu =?$

$$\nabla f(x^*) - \nabla h(x^*)\mu = 0 \tag{10.53}$$

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 \\ 2 \end{pmatrix} \mu = 0 \Leftrightarrow \mu = \frac{1}{2} \tag{10.54}$$

$x^* = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$, $\mu = \frac{1}{2}$ is a KKT point.

$$T_\Omega(x^*) = \mathcal{F}(x^*) = \{p \mid \nabla h^T p \geq 0\} = \{p \mid \begin{pmatrix} 0 \\ 2 \end{pmatrix}^T p \geq 0\} \tag{10.55}$$

$$C(x^*, \nabla) = \{p \mid \nabla h^T p = 0 \;\; \textit{if} \;\; \mu > 0\} \tag{10.56}$$

$$= \{p \mid \begin{pmatrix} 0 \\ 2 \end{pmatrix}^T p = 0\} \tag{10.57}$$

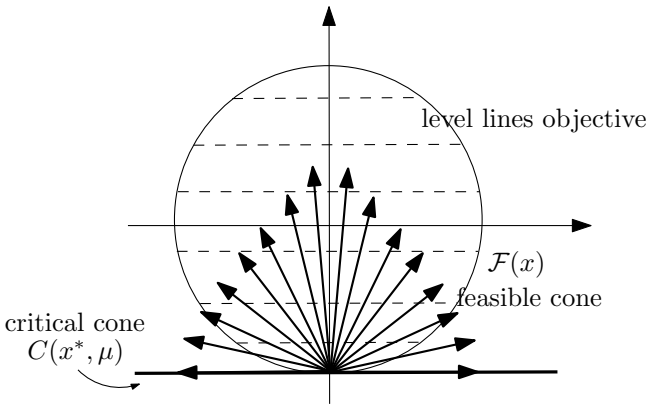


Figure 10.5: Conceptual visualization of Example 10.4.

Theorem 10.8 (SONC)**:** *Regard x^* with LICQ. If x^* is a local minimizer of the NLP, then:*

i) $\exists \lambda^*, \mu^*$ *so that KKT conditions hold;*

ii) $\forall p \in C(x^*, \mu^*)$ *holds that* $p^T \nabla_x^2 \mathcal{L}(x^*, \lambda^*, \mu^*) p \geq 0$

Theorem 10.9 (SOSC)**:** *If* x^* *satisfies LICQ and*

i) $\exists \lambda^*, \mu^*$ *so that KKT conditions hold;*

ii) $\forall p \in C(x^*, \mu^*),\ p \neq 0$, *holds that* $p^T \nabla_x^2 \mathcal{L}(x^*, \lambda^*, \mu^*) p > 0$

then x^* *is a strict local minimizer.*

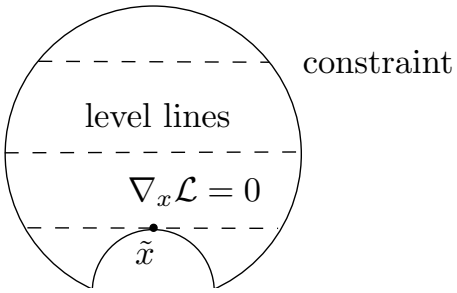


Figure 10.6: Motivation for Theorem 10.9: the point $\tilde{x}$ is not a local minimum.

Note $\nabla_x^2 \mathcal{L}(x^*, \lambda^*, \mu^*) = \nabla^2 f(x^*) - \sum \lambda_i \nabla^2 g_i(x^*) - \sum \mu_i \nabla^2 h_i(x^*)$, i.e. $\nabla_x^2 \mathcal{L}$ contains curvature of constraints.

Sketch of proof of both theorems
Regard the following restriction of the feasible set ($\bar{\Omega} \subset \Omega$):

$$\bar{\Omega} = \{x \mid g(x) = 0,\ h_i(x) = 0 \text{ if } i \in \mathcal{A}_+(x^*, \mu),\ h_i(x) \geq 0 \text{ if } i \in \mathcal{A}_0(x^*, \mu)\} \tag{10.58}$$

The critical cone is the tangent cone of this set $\bar{\Omega}$. First, for any feasible direction $p \in T_\Omega(x^*) \setminus C(x^*, \mu)$ we have $\nabla f(x^*)^T p > 0$. Thus, the difficult directions are those in the critical cone only. So let us regard points in the set $\bar{\Omega}$. For fixed λ, μ we have for all $x \in \bar{\Omega}$:

$$\mathcal{L}(x, \lambda, \mu) = f(x) - \sum \lambda_i \underbrace{g_i(x)}_{=0} - \sum_{i,\, \mu_i > 0} \mu_i \underbrace{h_i(x)}_{=0} - \underbrace{\sum_{i,\, \mu_i = 0} \mu_i h_i(x)}_{=0} \tag{10.59}$$

$$= f(x) \tag{10.60}$$

Also: $\nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = 0$. So for all $x \in \bar{\Omega}$ we have:

$$\begin{aligned} f(x) &= \mathcal{L}(x, \lambda, \mu) \\ &= \underbrace{\mathcal{L}(x^*, \lambda^*, \mu^*)}_{=f(x^*)} + \underbrace{\nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*)^T}_{=0}(x - x^*) + \frac{1}{2}(x - x^*)^T \nabla_x^2 \mathcal{L}(x^*, \lambda^*, \mu^*)(x - x^*) + o(\|x - x^*\|^2) \\ &= f(x^*) + \frac{1}{2}(x - x^*)^T \nabla_x^2 \mathcal{L}(x^*, \lambda^*, \mu^*)(x - x^*) + o(\|x - x^*\|^2) \end{aligned} \tag{10.61}$$

□

Example 10.5: Regard the example from before:

$$\mathcal{L}(x, \mu) = x_2 - \mu(1 - x_1^2 - x_2^2) \tag{10.62}$$

$$\nabla_x \mathcal{L} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \mu \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix} \tag{10.63}$$

$$\nabla_x^2 \mathcal{L} = 0 + \mu \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \tag{10.64}$$

For $\mu = \frac{1}{2}$ and $x^* = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$ we have:

$$C(x^*, \mu) = \{p \mid \nabla h^T p = 0\} = \{p \mid \begin{pmatrix} 0 \\ 2 \end{pmatrix}^T p = 0\} = \{\begin{pmatrix} p_1 \\ 0 \end{pmatrix}\} \tag{10.65}$$

$$p \in C \Rightarrow p = \begin{pmatrix} p_1 \\ 0 \end{pmatrix} \tag{10.66}$$

$$\nabla_x^2 \mathcal{L}(x^*, \lambda, \mu) = \frac{1}{2} \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{10.67}$$

SONC:

$$\underbrace{\begin{pmatrix} p_1 \\ 0 \end{pmatrix}^T \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ 0 \end{pmatrix}}_{=p_1^2} \geq 0 \tag{10.68}$$

SOSC:

$$\text{if } p \neq 0,\ p \in C \ :\ p^T \nabla_x^2 \mathcal{L} p > 0 \tag{10.69}$$

$$\text{if } p_1 \neq 0 \ :\ p_1^2 > 0 \tag{10.70}$$

Example 10.6:

$$\min x_2 \ \text{ s.t. } \ 2x_2 \geq x_1^2 - 1 - (x_2 + 1)^2 \tag{10.71}$$

Here $x^* = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$, $\mu = \frac{1}{2}$ is still a KKT point.

$$\nabla_x \mathcal{L}(x^*, \mu) = 0 \tag{10.72}$$

$$\nabla_x^2 \mathcal{L}(x^*, \mu) = \mu \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix} \tag{10.73}$$

Chapter 11

Equality Constrained Optimization Algorithms

In this chapter the problem to

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x) \tag{11.1a}$$

$$\text{subject to} \quad g(x) = 0 \tag{11.1b}$$

with $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$, where f and g are both smooth functions, will be further treated in detail.

11.1 Optimality Conditions

KKT Conditions

The necessary KKT optimality condition for

$$\mathcal{L}(x, \lambda) = f(x) - \lambda^T g(x) \tag{11.2}$$

leads to the expression

$$\nabla \mathcal{L}(x^*, \lambda^*) = 0 \tag{11.3}$$

$$g(x^*) = 0 \tag{11.4}$$

Keep in mind that this expression is only valid if we have LICQ, or equivalently stated, if the vectors $\nabla g_i(x^*)$ are lineary independent. Recall the definition of the gradient

$$\nabla g(x) = (\nabla g_1(x), \nabla g_2(x), \ldots, \nabla g_m(x)) \tag{11.5}$$

$$= \left(\frac{\partial g}{\partial x}(x) \right)^T. \tag{11.6}$$

The rank of the matrix $\nabla g(x^*)$ must be m to obtain LICQ. The tangent space is defined as

$$\begin{aligned} T_\Omega(x^*) &= \{p | \nabla g(x^*)^T p = 0\} && (11.7) \\ &= \text{kernel}(\nabla g(x^*)^T) && (11.8) \end{aligned}$$

An explicit form of kernel $\left(\nabla g(x)^T\right)$ can be obtained by a basis for this space $Z \in \mathbb{R}^{n\times(n-m)}$ such that the kernel$\left(\nabla g(x)^T\right) = \text{image}(Z)$, i.e. $\nabla g(x)^T Z = 0$ and $\text{rank}(Z) = n - m$. This basis $(Z_1 Z_2 \dots Z_{n-m})$ can be obtained by using a QR-factorization of the matrix $\nabla g(x)$.

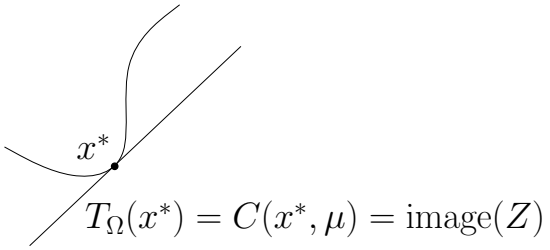


Figure 11.1: The critical cone equals the tangent cone when there are no inequality constraints.

SONC and SOSC

For equality constrained problems, SONC looks like

$$Z^T \nabla_x^2 \mathcal{L}(x^*, \lambda^*) Z \succcurlyeq 0 \tag{11.9}$$

The SOSC points out that if

$$Z^T \nabla_x^2 \mathcal{L}(x^*, \lambda^*) Z \succ 0 \tag{11.10}$$

and the LICQ and KKT conditions are satisfied, then x^* is a minimizer. The crucial role is played by the "reduced Hessian" $Z^T \nabla_x^2 \mathcal{L} Z$.

11.2 Equality Constrained QP

Regard the optimization problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2} x^T B x + g^T x && (11.11a) \\ \text{subject to} \quad & b + Ax = 0 && (11.11b) \end{aligned}$$

with $B \in \mathbb{R}^{n\times n}$, $A \in \mathbb{R}^{m\times n}$, $B = B^T$. The KKT condition leads to the equation

$$\begin{aligned} Bx + g - A^T \lambda &= 0 && (11.12a) \\ b + Ax &= 0. && (11.12b) \end{aligned}$$

In matrix notation

$$\begin{bmatrix} B & -A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = - \begin{bmatrix} g \\ b \end{bmatrix} \tag{11.13}$$

The left hand side matrix is nearly symmetric. With a few reformulations a symmetric matrix is obtained

$$\begin{bmatrix} B & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ -\lambda \end{bmatrix} = - \begin{bmatrix} g \\ b \end{bmatrix} \tag{11.14}$$

Lemma 11.1 (KKT-Matrix-Lemma)**:** *Define the matrix*

$$\begin{bmatrix} B & A^T \\ A & 0 \end{bmatrix} \tag{11.15}$$

as the KKT matrix. Regard some matrix $B \in \mathbb{R}^{n\times n}$, $B = B^T$, $A \in \mathbb{R}^{m\times n}$ with $m \leq n$. If the rank$(A) = m$ (A is of full rank, i.e. the LICQ holds) and for all $p \in$ kernel(A), $p \neq 0$ holds $p^T Bp > 0$ (SOSC). Then the KKT-matrix is invertible. (for the proof, we refer to [4] section 16.1)

Remark that for a QP

$$B = \nabla_x^2 \mathcal{L}(x^*, \lambda^*) \tag{11.16}$$
$$A = \nabla g(x)^T \tag{11.17}$$

so that the above invertibility condition is equivalent to SOSC. Note also that the QP is convex under these conditions.

11.2.1 Solving the KKT System

Solving KKT systems is an important research topic, there exist many ways to solve the system (11.12). Some methods are:

(i) Brute Force: obtain a dense LU-factorization of KKT-matrix

(ii) As the KKT-matrix is not definite, a standard Cholesky decomposition does not work. Use an indefinite Cholesky decomposition.

(iii) Schur complement method or so called "Range Space method": first eliminate x, by equation

$$x = B^{-1}(A^T\lambda - g) \tag{11.18}$$

and plug it in to the second equation (11.12b). Get λ from

$$b + A(B^{-1}(A^T\lambda - g)) = 0. \tag{11.19}$$

This method requires that B is invertible, which is not always true.

(iv) Null Space Method: First find basis $Z \in \mathbb{R}^{n\times(n-m)}$ of kernel(A), set $x = Zv + y$ with $b + Ay = 0$ (a special solution) every $x = Zv + y$ satisfies $b + Ax = 0$, so we have to regard only (11.12a). This is an unconstrained problem

$$\underset{v\,\in\,\mathbb{R}^{n-m}}{\text{minimize}} \quad g^T(Zv+y) + \frac{1}{2}(Zv+y)^T B(Zv+y) \tag{11.20a}$$

$$\Leftrightarrow \quad Z^T BZv + Z^T g + Z^T By = 0 \tag{11.20b}$$

$$\Leftrightarrow \quad v = -(Z^T BZ)^{-1}(Z^T g + Z^T By). \tag{11.20c}$$

The matrix $Z^T BZ$ is called "Reduced Hessian". This method is always possible if SOSC holds.

(v) Sparse direct methods like sparse LU decomposition.

(vi) Iterative methods of linear algebra.

11.3 Newton Lagrange Method

Regard again the optimization problem (11.1) as stated at the beginning of the chapter. The idea now is to apply Newton's method to solve the nonlinear KKT conditions

$$\nabla_x \mathcal{L}(x,\lambda) = 0 \tag{11.21a}$$

$$g(x) = 0 \tag{11.21b}$$

Define

$$\begin{bmatrix} x \\ \lambda \end{bmatrix} = w \text{ and } F(w) = \begin{bmatrix} \nabla_x \mathcal{L}(x,\lambda) \\ g(x) \end{bmatrix} \tag{11.22}$$

with $w \in \mathbb{R}^{n+m}$, $F : \mathbb{R}^{n+m} \to \mathbb{R}^{n+m}$, so that the optimization is just a nonlinear root finding problem

$$F(w) = 0, \tag{11.23}$$

which we solve again by Newton's method.

$$F(\omega_k) + \frac{\partial F}{\partial w_k}(w_k)(w - w_k) = 0 \tag{11.24}$$

Written in terms of gradients

$$\nabla_x \mathcal{L}(x_k,\lambda_k) + \nabla_x^2 \mathcal{L}(x,\lambda)(x - x_k) - \nabla g(x_k)(\lambda - \lambda_k) = 0 \tag{11.25}$$

$\nabla_x^2 \mathcal{L}(x,\lambda)(x - x_k)$ is the linearisation with respect to x, $\nabla g(x_k)(\lambda - \lambda_k)$ the linearisation with respect to λ. Recall that $\nabla \mathcal{L} = \nabla f - \nabla g \lambda$.

$$g(x_k) + \nabla g(x_k)^T (x - x_k) = 0 \tag{11.26}$$

Written in matrix form an interesting result is obtained

$$\begin{bmatrix} \nabla_x \mathcal{L} \\ g \end{bmatrix} + \underbrace{\begin{bmatrix} \nabla_x^2 \mathcal{L} & \nabla g \\ \nabla g^T & 0 \end{bmatrix}}_{\text{KKT-matrix}} \begin{bmatrix} x - x_k \\ -(\lambda - \lambda_k) \end{bmatrix} = 0 \tag{11.27}$$

The KKT-matrix is invertible if the KKT-matrix lemma holds. From this point it is clear that at a given solution (x^*, λ^*) with LICQ and SOSC, the KKT-matrix would be invertible. This also holds in the neighborhood of (x^*, λ^*). Thus, if (x^*, λ^*) satisfies LICQ and SOSC then the Newton method is well defined for all (x_0, λ_0) in neighborhood of (x^*, λ^*) and converges Q-quadratically.

The method is stated as an algorithm in Algorithm 7.

Algorithm 7 Equality constrained Newton Lagrange method

Choose: x_0, λ_0, ϵ
Set: $k = 0$

while norm $\begin{bmatrix} \nabla \mathcal{L}(x_k, \lambda_k) \\ g(x^*) \end{bmatrix} \geq \epsilon$ **do**
  get Δx_k and $\Delta \lambda_k$ from (11.30)
  $x_{k+1} = x_k + \Delta x_k$
  $\lambda_{k+1} = \lambda_k + \Delta \lambda_k$
  $k = k + 1$
end while

Using the definition

$$\lambda_{k+1} = \lambda_k + \Delta \lambda_k \tag{11.28}$$
$$\nabla \mathcal{L}(x_k, \lambda_k) = \nabla f(x_k) - \nabla g(x_k) \lambda_k \tag{11.29}$$

the system (11.27) needed for calculating $\Delta \lambda_k$ and Δx_k is equivalent to

$$\begin{bmatrix} \nabla f(x_k) \\ g(x_k) \end{bmatrix} + \begin{bmatrix} \nabla^2 \mathcal{L} & \nabla g \\ \nabla g^T & 0 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ -\lambda_{k+1} \end{bmatrix} = 0. \tag{11.30}$$

This formulation shows that the new iterate does not depend strongly on the old multiplier guess, only via the Hessian matrix. We will later see that we can approximate the Hessian with different methods.

11.4 Quadratic Model Interpretation

Theorem 11.2: *x_{k+1} and λ_{k+1} are obtained from the solution of a QP:*

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \nabla f(x_k)^T (x - x_k) + \frac{1}{2} (x - x_k)^T \nabla^2 \mathcal{L}(x_k, \lambda_k)(x - x_k) \tag{11.31a}$$

$$\text{subject to} \quad g(x_k) + \nabla g(x_k)^T (x - x_k) = 0 \tag{11.31b}$$

So we can get a QP solution x^{QP} and λ^{QP} and take it as next NLP solution guess x_{k+1} and λ_{k+1}.

Proof. KKT of QP

$$\nabla f(x_k) + \nabla^2 \mathcal{L}(x_k, \lambda_k)(x^{\mathrm{QP}} - x_k) - \nabla g(x_k)\lambda^{\mathrm{QP}} = 0 \tag{11.32}$$

$$g + \nabla g^T (x^{\mathrm{QP}} - x_k) = 0 \tag{11.33}$$

□

More generally, one can replace $\nabla_x^2 \mathcal{L}(x_k, \lambda_k)$ by some approximation B_k, ($B_k = B_k^T$ often $B_k \succcurlyeq 0$) by Quasi-Newton updates or other.

11.5 Constrained Gauss-Newton

Regard:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\|F(x)\|_2^2 \tag{11.34a}$$

$$\text{subject to} \quad g(x) = 0 \tag{11.34b}$$

As in the unconstrained case, linearize both F and g. Get approximation by

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\|F(x_k) + J(x_k)(x - x_k)\|_2^2 \tag{11.35a}$$

$$\text{subject to} \quad g(x_k) + \nabla g(x_k)^T (x - x_k) = 0 \tag{11.35b}$$

This is a LS-QP which is convex. We call this the constrained Gauss-Newton method, this approach gets new iterate x_{k+1} by solution of (11.35a)–(11.35b) in each iteration. Note that no multipliers λ_{k+1} are needed. The KKT conditions of LS-QP

$$\nabla_x \frac{1}{2}\|F + J(x - x_k)\|_2^2 \;=\; J^T J(x - x_k) + J^T F \tag{11.36}$$

equals

$$J^T J(x - x_k) + J^T F - \nabla g \lambda \;=\; 0 \tag{11.37}$$

$$g + \nabla g^T (x - x_k) \;=\; 0 \tag{11.38}$$

Recall that $J^T J$ the same is as by Newton iteration, but we replace the Hessian. The constrained Gauss-Newton gives a Newton type iteration with $B_k = J^T J$. For LS,

$$\nabla_x^2 \mathcal{L}(x, \lambda) \;=\; J(x)^T J(x) + \sum F_i(x) \nabla^2 F_i(x) - \sum \lambda_i \nabla^2 g_i(x) \tag{11.39}$$

One can show that $\|\lambda\|$ gets small if $\|F\|$ is small. As in the unconstrained case, CGN converges well if $\|F\| \approx 0$.

11.6 An Equality Constrained BFGS Method

Regard the equality constrained BFGS method, as stated in algorithm 8.

11.7 Local Convergence

Theorem 11.3 (Newton type convergence)**:** *Regard the root finding problem*

$$F(x) = 0, \quad F : \mathbb{R}^n \to \mathbb{R}^n \tag{11.40}$$

with x^* satisfying $F(x^*) = 0$ a local solution, $J(x) = \frac{\partial F}{\partial x}(x)$, and iteration $x_{k+1} = x_k - M_k^{-1}F(x_k)$ with $\forall k : M_k \in \mathbb{R}^{n\times m}$ invertible, and a Lipschitz condition

$$\|M_k^{-1}(J(x_k) - J(x^*))\| \leq \omega\|x_k - x^*\| \tag{11.41}$$

and a compatibility condition with $\kappa < 1$:

$$\|M_k^{-1}(J(x_k) - M_k)\| \leq \kappa_k < \kappa \quad \text{and} \quad \|x_0 - x^*\| \leq \frac{2}{\omega}(1-\kappa) \tag{11.42}$$

then $x_k \to x^*$ with linear rate or even quadratic rate if $\kappa = 0$ or superlinear rate if $\kappa_k \to 0$ (proof as before).

Corollary: *Newton-type constrained optimization converges*

- *quadratically if $B_k = \nabla^2\mathcal{L}(x_k, \lambda_k)$,*
- *superlinearly if $B_k \to \nabla^2\mathcal{L}(x_k, \lambda_k)$ (BFGS),*
- *linearly if $\|B_k - \nabla^2\mathcal{L}(x_k, \lambda_k)\|$ is not too big (Gauss-Newton).*

Proof.

$$J_k = \begin{bmatrix} \nabla^2\mathcal{L}(x_k, \lambda_k) & -\frac{\partial g}{\partial x}(x_k)^T \\ \frac{\partial g}{\partial x}(x_k) & 0 \end{bmatrix} \tag{11.43}$$

$$M_k = \begin{bmatrix} B_k & -\frac{\partial g}{\partial x}(x_k)^T \\ \frac{\partial g}{\partial x}(x_k) & 0 \end{bmatrix} \tag{11.44}$$

$$J_k - M_k = \begin{bmatrix} \nabla^2\mathcal{L}(x_k, \lambda_k) - B_k & 0 \\ 0 & 0 \end{bmatrix} \tag{11.45}$$

□

Algorithm 8 Equality constrained BFGS method

Choose x_0, B_0, tolerance
k = 0
Evaluate $\nabla f(x_0)$, $g(x_0)$, $\frac{\partial g}{\partial x}(x_0)$

while $\|g(x_k)\| >$ tolerance or $\|\nabla \mathcal{L}(x_k, \widetilde{\lambda}_k)\| >$ tolerance **do**
Solve KKT-system:
$$\begin{bmatrix} \nabla f \\ g \end{bmatrix} + \begin{bmatrix} B_k & \frac{\partial g}{\partial x}^T \\ \frac{\partial g}{\partial x} & 0 \end{bmatrix} \begin{bmatrix} p_k \\ -\widetilde{\lambda}_k \end{bmatrix} = 0$$
Set $\Delta\lambda_k = \widetilde{\lambda}_k - \lambda_k$

Choose step length $t_k \in (0, 1]$ (details 11.7)

$x_{k+1} = x_k + t_k p_k$

$\lambda_{k+1} = \lambda_k + t_k \Delta\lambda_k$

Compute old Lagrange gradient:
$\nabla_x \mathcal{L}(x_k, \lambda_{k+1}) = \nabla f(x_k) - \frac{\partial g}{\partial x}(x_k)^T \lambda_{k+1}$

Evaluate $\nabla f(x_{k+1})$, $g(x_{k+1})$, $\frac{\partial g}{\partial x}(x_{k+1})$

Compute new Lagrange gradient $\nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1})$

Set $s_k = x_{k+1} - x_k$

Set $y_k = \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1})$

Calculate B_{k+1} (e.g. with a BFGS update) using s_k and y_k.

$k = k + 1$
end while

Remark: B_{k+1} can alternatively be obtained by either calculating the exact Hessian $\nabla^2 \mathcal{L}(x_{k+1}, \lambda_{k+1})$ or by calculating the Gauss-Newton Hessian $(J(x_{k+1})^T J(x_{k+1})$ for a LS objective function).

Note that we could still ensure convergence even if the Jacobians $\frac{\partial g}{\partial x}$ were approximated. This could lead to potentially cheaper iterations as building and factoring the KKT matrix is the main cost per iteration. As in all Newton-type methods, we only need to ensure that the residual $F(x)$ is exactly evaluated. The Lagrange gradient can be obtained by reverse automatic differentiation without ever evaluating $\frac{\partial g}{\partial x}$.

11.8 Globalization by Line Search

Idea: use "merit function" to measure progress in both *objective* and *constraints*.

Definition 11.1 (L_1-merit function)
the "L_1-merit function" is defined to be $T_1(x) = f(x) + \sigma\|g(x)\|_1$ with $\sigma > 0$.

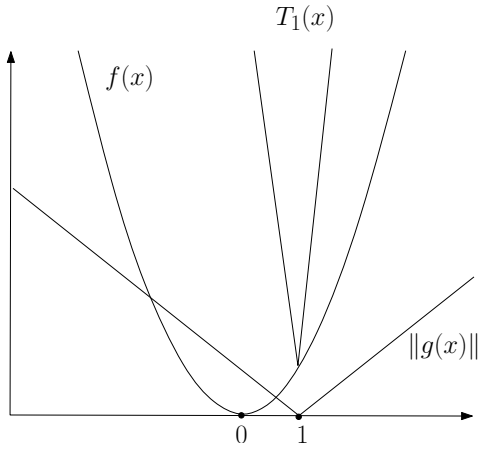


Figure 11.2: An example of a L1 merit function with $f(x) = x^2$, $g(x) = x - 1$ and $\sigma = 10$.

Definition 11.2 (directional derivative)
the "directional derivative of F at x in direction p" is $DF(x)[p] = \lim_{t \to 0, t > 0} \frac{F(x+tp) - F(x)}{t}$.

Example 11.1 (directional derivative)**:**

$$\begin{aligned} F(x) &= |x-1| && (11.46)\\ DF(1)[2] &= \lim_{t\to 0, t>0} \frac{|1+t\cdot 2-1|-|1-1|}{t} = 2 && (11.47)\\ DF(1)[-3] &= \lim_{t\to 0, t>0} \frac{|1+t\cdot(-3)-1|-|1-1|}{t} = 3 && (11.48)\\ & && (11.49) \end{aligned}$$

Lemma 11.4: *If* p *&* $\widetilde{\lambda}$ *solve* $\begin{bmatrix}\nabla f\\ g\end{bmatrix} + \begin{bmatrix} B & \frac{\partial g}{\partial x}^T \\ \frac{\partial g}{\partial x} & 0\end{bmatrix}\begin{bmatrix} p \\ -\widetilde{\lambda}\end{bmatrix} = 0$ *then*

$$\begin{aligned} DT_1(x)[p] &= \nabla f(x)^T p - \sigma\|g(x)\|_1 && (11.50)\\ DT_1(x)[p] &\leq -p^T B p - (\sigma - \|\widetilde{\lambda}\|_\infty)\|g(x)\|_1 && (11.51)\\ & && (11.52) \end{aligned}$$

Corollary: *If* $B \succ 0$ *&* $\sigma \geq \|\widetilde{\lambda}\|_\infty$ *then* p *is a descent direction of* T_1*.*

Proof of the lemma.

$$\begin{aligned} T_1(x+tp) &= f(x+tp) + \sigma\|g(x+tp)\|_1 && (11.53)\\ &= f(x) + t\nabla f(x)^T p + \sigma\|g(x) + \frac{\partial g}{\partial x}(x)pt\|_1 + O(t^2) && (11.54)\\ &= f(x) + t\nabla f(x)^T p + \sigma\|g(x)(1-t)\|_1 + O(t^2) && (11.55)\\ &= f(x) + t\nabla f(x)^T p + \sigma(1-t)\|g(x)\|_1 + O(t^2) && (11.56)\\ &= T_1(x) + t(\nabla f(x)^T p - \sigma\|g(x)\|_1) + O(t^2) && (11.57)\\ &\Rightarrow (11.50) && (11.58) \end{aligned}$$

$$\begin{aligned} \nabla f(x) + Bp - \frac{\partial g}{\partial x}(x)^T\widetilde{\lambda} &= 0 && (11.59)\\ \nabla f(x)^T p &= \widetilde{\lambda}^T\frac{\partial g}{\partial x}(x)p - p^T B p && (11.60)\\ &= -\widetilde{\lambda}^T g(x) - p^T B p && (11.61)\\ |\nabla f(x)^T p| &\leq \|\widetilde{\lambda}\|_\infty\|g(x)\|_1 - p^T B p && (11.62)\\ &\Rightarrow (11.50)\Rightarrow(11.51) && (11.63) \end{aligned}$$

□

In Algorithm 8 use Armijo backtracking with L_1-merit function, ensure $\sigma \geq \|\widetilde{\lambda}\|_\infty$ (if not, increase σ).

11.9 Careful BFGS Updating

How can we make sure that B_k remains positive definite?

Lemma 11.5: *If $B_k \succ 0$ and $y_k^T s_k > 0$ then B_{k+1} from BFGS update is positive definite.*

Proof. [4] page 137-138. □

This is as good as we can desire because:

Lemma 11.6: *If $y_k^T s_k < 0$ & $B_{k+1} s_k = y_k$ then B_{k+1} is not positive semidefinite.*

Proof. $s_k^T B_{k+1} s_k = s_k^T y_k < 0$ i.e. s_k is a direction of negative curvature of B_{k+1}. □

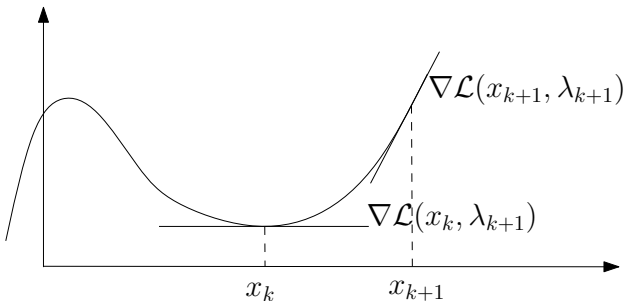


Figure 11.3: Visualization of Lemma 11.6. Remark that $y_k = \nabla\mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla\mathcal{L}(x_k, \lambda_{k+1})$ and $s_k = x_{k+1} - x_k$.

Powell's trick: If $y_k^T s_k < 0.2 s_k^T B s_k$ then do update with a $\widetilde{y_k}$ instead of y_k with $\widetilde{y_k} = y_k + \theta(B_k s_k - y_k)$ so that $\widetilde{y_k}^T s_k = 0.2 s_k^T B_k s_k > 0$.

The explicit formula for θ is easily seen to be

$$\theta = \begin{cases} \frac{0.2 s_k^T B_k s_k - s_k^T y_k}{s_k^T B_k s_k - s_k^T y_k} & \text{if } y_k^T s_k < 0.2 s_k^T B_k s_k \\ 0 & \text{else} \end{cases}$$

Remark (1). If $\theta = 1$ then $\widetilde{y_k} = B_k s_k$ and $B_{k+1} = B_k$. Thus, the choice of θ between 0 and 1 damps the BFGS update.

Remark (2). Note that the new Hessian B_{k+1} will satisfy the modified secant condition $B_{k+1} s_k = \tilde{y}_k$, so we will have $s_k^T B_{k+1} s_k = s_k^T \tilde{y}_k > 0.2 s_k^T B_k s_k$. The damping thus ensures that the positive

curvature of the Hessian in direction s_k, which is expressed in the term $s_k^T B_k s_k$, will never decrease by more than a factor 5.

Chapter 12

Inequality Constrained Optimization Algorithms

For simplicity, drop equalities and regard:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x) \tag{12.1}$$

$$\text{subject to} \quad h(x) \geq 0 \tag{12.2}$$

In the KKT conditions we had (for $i = 1, \ldots, q$):

1. $\nabla f(x) - \sum_{i=1}^{q} \nabla h_i(x) \mu_i = 0$
2. $h_i(x) \geq 0$
3. $\mu_i \geq 0$
4. $\mu_i h_i(x) = 0$

Conditions 2, 3 and 4 are non-smooth, which implies that Newton's method will not work here.

12.1 Quadratic Programming via Active Set Method

Regard the QP problem to be solved:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad g^T x + \frac{1}{2} x^T B x \tag{12.3}$$

$$\text{subject to} \quad Ax + b \geq 0 \tag{12.4}$$

Assume a convex QP ($B \succeq 0$). The KKT conditions are necessary and sufficient for global optimality (this is the basis for the algorithm):

$$\begin{aligned} Bx^* + g - A^T\mu^* &= 0 && (12.5)\\ Ax^* + b &\geq 0 && (12.6)\\ \mu^* &\geq 0 && (12.7)\\ \mu_i^*(Ax^* + b)_i &= 0 && (12.8) \end{aligned}$$

for $i = 1, \ldots, q$. How do we find x^*, μ^* and the corresponding active set $\mathcal{A}(x^*) \subset \{1, \ldots, q\}$ so that KKT holds?

Definition 12.1 (Index Set)

$$\begin{aligned} \mathbb{A} &\subset \{1, \ldots, q\} \text{ “Active”} && (12.9)\\ \mathbb{I} &= \{1, \ldots, q\} \backslash \mathbb{A} \text{ “Inactive”} && (12.10) \end{aligned}$$

Vector division

$$b = \begin{pmatrix} b_{\mathbb{A}} \\ b_{\mathbb{I}} \end{pmatrix} \quad b \in \mathbb{R}^q \tag{12.11}$$

Matrix division

$$A = \begin{pmatrix} A_{\mathbb{A}} \\ A_{\mathbb{I}} \end{pmatrix} \tag{12.12}$$

ie

$$Ax + b \geq 0 \Longleftrightarrow A_{\mathbb{A}}x + b_{\mathbb{A}} \geq 0 \text{ AND } A_{\mathbb{I}}x + b_{\mathbb{I}} \geq 0 \tag{12.13}$$

$$\tag{12.14}$$

Lemma 12.1: *x^* is a global minimizer of the QP iff there exist an index set $\mathbb{A}$ and $\mathbb{I}$ and a vector $\mu_{\mathbb{A}}^*$ so that:*

$$\begin{aligned} Bx^* + g - A_{\mathbb{A}}^T\mu_{\mathbb{A}}^* &= 0 && (12.15)\\ A_{\mathbb{A}}x^* + b_{\mathbb{A}} &= 0 && (12.16)\\ A_{\mathbb{I}}x^* + b_{\mathbb{I}} &\geq 0 && (12.17)\\ \mu_{\mathbb{A}}^* &\geq 0 && (12.18) \end{aligned}$$

and

$$\mu^* = \begin{pmatrix} \mu_{\mathbb{A}}^* \\ \mu_{\mathbb{I}}^* \end{pmatrix} \quad \textit{with} \quad \mu_{\mathbb{I}}^* = 0 \tag{12.19}$$

The *active set method* idea and the *primal active set method* idea are shown in algorithm 9 and 10.

Algorithm 9 Active set method idea

Choose a set $\mathbb{A}$
Solve (12.15) and (12.16) to get x^* and μ^*

if (12.17) and (12.18) are satisfied **then**
  Solution found
else
  Change set $\mathbb{A}$ by adding or removing constraint indices
end if

For the last step many variants exists: primal, dual, primal-dual, online... E.g., QPSOL, quadprog (Matlab) and qpOASES.

Algorithm 10 Primal active set method in detail

Choose a feasible starting point x_0 with corresponding active set $\mathbb{A}_0$
$k \leftarrow 0$

while *no solution found* **do**
  Solve $B\tilde{x}_k + g - A_{\mathbb{A}_k}^T \tilde{\mu}_k = 0$ and $A_{\mathbb{A}_k}\tilde{x}_k + b_{\mathbb{A}_k} = 0$
  Go on a line from x_k to $\tilde{x}_k$: $x_{k+1} = x_k + t_k(\tilde{x}_k - x_k)$ with some $t_k \in [0, 1]$ so that x_{k+1} is feasible

  if $t_k < 1$ **then**

    $\mathbb{A}_{k+1} \leftarrow \mathbb{A}_k \cup \{i^*\}$ *(Add a blocking constraint i^* to $\mathbb{A}$)*
    $k \leftarrow k + 1$

  else if $t_k = 1$ **then**
    ($\tilde{x}_k$ is feasible)

    if $\tilde{\mu}_k \geq 0$ **then**
      Solution found
    else
      Drop index i^{**} in $\mathbb{A}_k$ with $\tilde{\mu}_{k,i^{**}} < 0$ and $\mathbb{A}_{k+1} = \mathbb{A}_k \backslash \{i^{**}\}$
      $k \leftarrow k + 1$
    end if

  end if
end while

Remark: we can prove that $f(x_{k+1}) \leq f(x_k)$ (with f the quadratic performance index).

Example 12.1 (Active set method)**:** Consider the problem

$$\begin{aligned} \min \quad & \|x\|_2^2 && (12.20)\\ \text{subject to} \quad & x_1 \geq 1 && (12.21)\\ & x_2 + 1 \geq 0 && (12.22)\\ & 1 - x_2 \geq 0 && (12.23) \end{aligned}$$

We choose x_0 as a feasible starting point with corresponding active set $\mathbb{A}_0 = \{3\}$. At the first iteration, the infeasible point $\tilde{x}_0$ is obtained by solving the two equations. This point will be avoided by adding second constraint (1 on Figure 12.1) as a blocking constraint because $t_0 < 1$. The new iterate is x_1 with active set $\mathbb{A}_1 = \{1, 3\}$. For the second iteration, by solving the equations we get $\tilde{x}_1 = x_1$ and $t_k = 1$. Regarding the negative multiplier $\tilde{\mu}_{k,3}$ we drop index 3 in $\mathbb{A}_1$ and get $\mathbb{A}_2 = \{1\}$. The next iteration has as conclusion $\tilde{x}_2 = x^*$ and is the last iteration.

It can be proven that $f(x_{k+1}) < f(x_k)$ in each iteration.

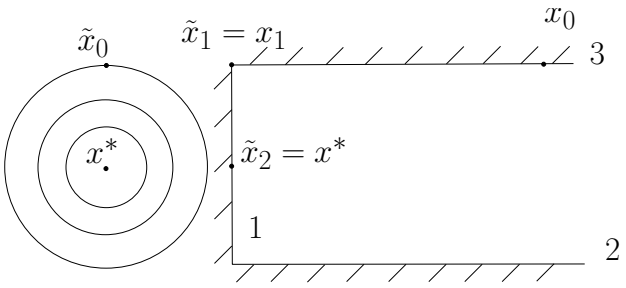


Figure 12.1: Visualization of Example 12.1.

12.2 Sequential Quadratic Programming (SQP)

Regard the NLP:

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) && (12.24)\\ \text{subject to} \quad & h(x) \geq 0 && (12.25) \end{aligned}$$

The SQP idea is to solve in each iteration the QP:

$$\begin{aligned} \underset{p}{\text{minimize}} \quad & \nabla f(x_k)^T p + \frac{1}{2} p^T B p && (12.26)\\ \text{subject to} \quad & h(x_k) + \frac{\partial h}{\partial x}(x_k) p \geq 0 && (12.27) \end{aligned}$$

Local convergence would follow from equality constrained optimization if the active set of the QP is the same as the active set of the NLP, at least in the last iterations.

Theorem 12.2 (Robinson)**:** *If x^* is a local minimizer of the NLP with LICQ and strict complementarity and if x_k is close enough to x^* and $B \succeq 0$ and B is positive definite on the nullspace of the linearization of the active constraints, then the solution of the QP has the same active set as the NLP.*

Proof. Define $\mathbb{A} = \mathcal{A}(x^*)$ and regard:

$$\nabla f(x) + Bp - \frac{\partial h_{\mathbb{A}}}{\partial x}(x)^T \mu_{\mathbb{A}}^{\mathrm{QP}} = 0 \tag{12.28}$$

$$h_{\mathbb{A}}(x) + \frac{\partial h_{\mathbb{A}}}{\partial x}(x)p = 0 \tag{12.29}$$

this defines an implicit function

$$\begin{pmatrix} p(x,B) \\ \mu_{\mathbb{A}}^{\mathrm{QP}}(x,B) \end{pmatrix} \tag{12.30}$$

with

$$p(x^*,B) = 0 \text{ and } \mu_{\mathbb{A}}^{\mathrm{QP}}(x^*,B) = \mu_{\mathbb{A}}^*) \tag{12.31}$$

This follows from

$$\nabla f(x^*) + Bp - \frac{\partial h_{\mathbb{A}}}{\partial x}(x^*)^T \mu_{\mathbb{A}}^* = 0 \Longleftrightarrow \nabla_x \mathcal{L}(x^*,\mu^*) = 0 \tag{12.32}$$

$$h_{\mathbb{A}}(x^*) + \frac{\partial h_{\mathbb{A}}}{\partial x}(x^*)0 = 0 \tag{12.33}$$

which hold because of

$$h_{\mathbb{A}}(x^*) = 0 \tag{12.34}$$

$$h_{\mathbb{I}}(x^*) > 0 \tag{12.35}$$

$$\mu_{\mathbb{I}}^* = 0 \tag{12.36}$$

Note that $\mu_{\mathbb{A}}^* > 0$ because of strict complementarity.

For x close to x^*, due to continuity of $p(x,B)$ and $\mu_{\mathbb{A}}^{\mathrm{QP}}(x,B)$ we still have $h_{\mathbb{I}}(x) > 0$ and

$$\mu_{\mathbb{A}}^{\mathrm{QP}}(x,B) > 0 \tag{12.37}$$

and even more:

$$h_{\mathbb{I}}(x) + \frac{\partial h_{\mathbb{I}}}{\partial x}(x)p(x,B) > 0 \tag{12.38}$$

Therefore a solution of the QP has the same same active set as the NLP and also satisfies strict complementarity. □

Remark. We can generalise his Theorem to the case where the jacobian $\frac{\partial h}{\partial x}(x_h)$ is only approximated.

12.3 Powell's Classical SQP Algorithm

For an equality and inequality constrained NLP, we can use the BFGS algorithm as before but:

1. We solve an *inequality constrained* QP instead of a linear system
2. We use $T_1(x) = f(x) + \sigma\|g(x)\|_1 + \sigma\sum_{i=1}^{q}|\min(0, h_i(x))|$
3. Use full Lagrange gradient $\nabla_x\mathcal{L}(x, \lambda, \mu)$ in the BFGS formula

(eg "fmincon" in Matlab).

12.4 Interior Point Methods

The IP method is an alternative for the active set method for QPs or LPs or for the SQP method. The previous methods had problems with the non-smoothness in the KTT-conditions (2), (3) and (4) (for $i = 1, \ldots, q$):

1. $\nabla f(x) - \sum_{i=1}^{q}\nabla h_i(x)\mu_i = 0$
2. $h_i(x) \geq 0$
3. $\mu_i \geq 0$
4. $\mu_i h_i(x) = 0$.

The IP-idea is to replace 2,3 and 4 by a smooth condition (which is an approximation): $h_i(x)\mu_i = \tau$ with $\tau > 0$ small. The KKT-conditions now become a smooth root finding problem:

$$\nabla f(x) - \sum_{i=1}^{q}\nabla h_i(x)\mu_i = 0 \tag{12.39}$$

$$h_i(x)\mu_i - \tau = 0 \quad i = 1, \ldots, q \tag{12.40}$$

These conditions are called the *IP-KKT conditions* and can be solved by Newtons method and yield solutions $\bar{x}(\tau)$ and $\bar{\mu}(\tau)$.

We can show that for $\tau \to 0$

$$\bar{x}(\tau) \to x^* \tag{12.41}$$

$$\bar{\mu}(\tau) \to \mu^* \tag{12.42}$$

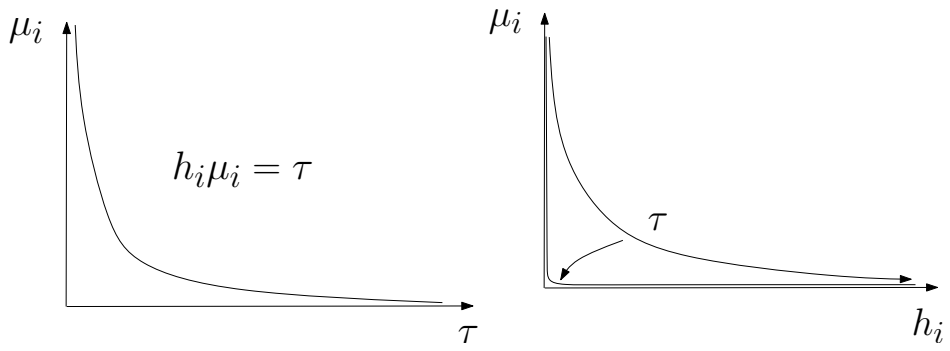


Figure 12.2: The interior point method idea: make the KKT-conditions a smooth root finding problem.

The IP algorithm:

1. Start with a big $\tau \gg 0$, choose $\beta \in (0, 1)$
2. Solve IP-KKT to get $\bar{x}(\tau)$ and $\bar{\mu}(\tau)$
3. Replace $\tau \leftarrow \beta\tau$ and go to 2.
   (initialize Newton iteration with old solution).

Remark (1). The set of solutions $\begin{pmatrix} \bar{x}(\tau) \\ \bar{\mu}(\tau) \end{pmatrix}$ for $\tau \in (0, \infty)$ is called the *central path*.

Remark (2). In fact, the IP-KKT is equivalent to FONC of the *Barrier Problem* (BP):

$$\min_x f(x) - \tau \sum_{i=1}^{q} \log h_i(x) \tag{12.43}$$

$$\text{FONC of BP} \iff \nabla f(x) - \tau \sum_{i=1}^{q} \frac{1}{h_i(x)} \nabla h_i(x) = 0 \tag{12.44}$$

with $\mu_i = \frac{\tau}{h_i(x)}$ this is equivalent to IP-KKT.

Example 12.2: (Barrier Problem) The problem

$$\text{minimise} \quad x \tag{12.45}$$

$$\text{subject to} \quad x \geq 0 \tag{12.46}$$

could be translated into a Barrier problem:

$$\text{minimise} \quad x - \tau \log(x) \tag{12.47}$$

with visualization given in Figure 12.3.

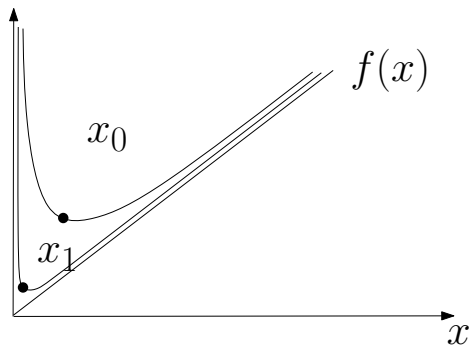


Figure 12.3: Visualization of Example 12.2.

Optimization software based on interior point methods: For convex problems, IP methods are well understood with strong complexity results. For LPs and QPs and other convex problems, the IP method is successfully implemented e.g. in OOQP, CPLEX, SeDuMi, SDPT3, CVX, or CVXGEN. But IP methods also exist for general nonlinear programs where they still work very reliable. A very powerful and widely used IP method for sparse NLP is the open-source code IPOPT.

Chapter 13

Optimal Control Problems

We regard a dynamical system with dynamics

$$x_{k+1} \quad = \quad f(x_k, u_k) \tag{13.1}$$

with u_k the "controls" or "inputs" and x_k the "states". Let $x_k \in \mathbb{R}^{n_x}$ and let $u_k \in \mathbb{R}^{n_u}$ with $k = 0, \ldots, N-1$.

If we know the initial state x_0 and the controls $u_0, \ldots, u_{N-1}$ we could simulate the system to obtain all other states. In optimization, we might have different requirements than just a fixed initial state. We might, for example, have both a fixed initial state and a fixed terminal state that we want to reach. Or we might just look for periodic sequences with $x_0 = x_N$. All these desires on the initial and the terminal state can be expressed by a boundary constraint function

$$r(x_0, x_N) = 0. \tag{13.2}$$

For the case of fixed initial value, this function would just be

$$r(x_0, x_N) \quad = \quad x_0 - \bar{x}_0 \tag{13.3}$$

where $\bar{x}_0$ is the fixed initial value and not an optimization variable. Another example would be to have both ends fixed, resulting in a function r of double the state dimension, namely:

$$r(x_0, x_N) \quad = \quad \begin{bmatrix} x_0 - \bar{x}_0 \\ x_N - \bar{x}_N \end{bmatrix}. \tag{13.4}$$

Finally, periodic boundary conditions can be imposed by setting

$$r(x_0, x_N) \quad = \quad (x_0 - x_N). \tag{13.5}$$

An illustration of inputs and states is given in Figure 13.1.

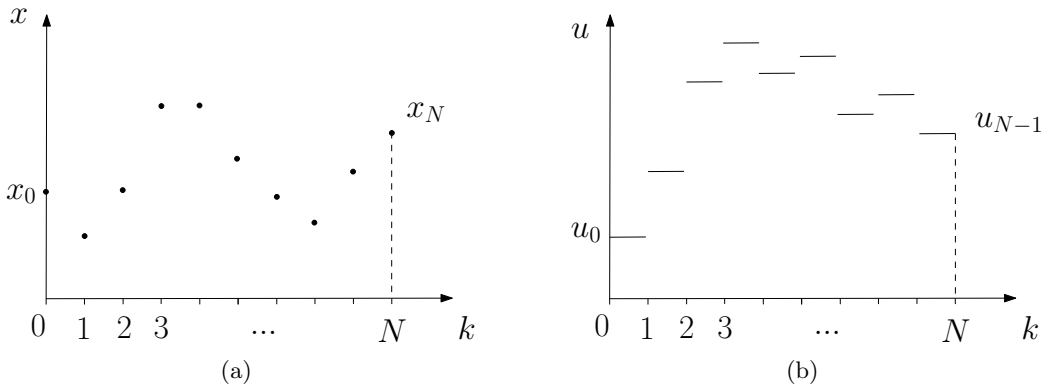


Figure 13.1: A conceptual example of an optimal control problem with states (a) and controls (b).

13.1 Optimal Control Problem (OCP) Formulation

The simplified optimal control problem in discrete time that we regard in this chapter is the following equality constrained NLP.

$$\begin{aligned} \underset{x_0, u_0, x_1, \ldots, u_{N-1}, x_N}{\text{minimize}} \quad & \sum_{k=0}^{N-1} L(x_k, u_k) + E(x_N) && \text{(13.6a)} \\ \text{subject to} \quad & x_{k+1} - f(x_k, u_k) = 0 \quad \text{for} \quad k = 0, \ldots, N-1 && \text{(13.6b)} \\ & r(x_0, x_N) = 0 && \text{(13.6c)} \end{aligned}$$

Note that (13.6b) contains many constraints. Other constraints that we just omit for notational simplicity could be inequalities of the form

$$h(x_k, u_k) \quad \geq \quad 0, \; k = 0, \ldots, N-1 \tag{13.7}$$

We also remark that any free parameter p could be added to the optimisation formulation above, e.g. the constant size of a vessel in a chemical reactor. For this we could define an extra dummy state for $k = 0, \ldots, N-1$

$$p_{k+1} \quad = \quad p_k. \tag{13.8}$$

13.2 KKT Conditions of Optimal Control Problems

First summarize the variables $w = \{x_0, u_0, x_1, u_1, \ldots, u_{N-1}, x_N\}$ and summarize the multipliers $\lambda = \{\lambda_1, \ldots, \lambda_N, \lambda_r\}$. The optimal control problem has the form

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & F(w) && \text{(13.9a)} \\ \text{subject to} \quad & G(w) \quad = \quad 0 && \text{(13.9b)} \end{aligned}$$

Where

$$G(w) = \begin{bmatrix} x_1 - f(x_0, u_0) \\ x_2 - f(x_1, u_1) \\ \vdots \\ x_N - f(x_{N-1}, u_{N-1}) \\ r(x_0, x_N) \end{bmatrix} \tag{13.9c}$$

The Lagrangian function has the form

$$\begin{aligned} \mathcal{L}(w, \lambda) &= F(w) - \lambda^T G(w) \\ &= \sum_{k=0}^{N-1} L(x_k, u_k) + E(x_N) - \sum_{k=0}^{N-1} \lambda_{k+1}^T (x_{k+1} - f(x_k, u_k)) \\ &\quad -\lambda_r^T r(x_0, x_N) \end{aligned} \tag{13.10}$$

The KKT-conditions of the problem are

$$\nabla_w \mathcal{L}(w, \lambda) = 0 \tag{13.11a}$$
$$G(w) = 0 \tag{13.11b}$$

In more detail, the derivative of $\mathcal{L}$ with respect to x_k, where $k = 0$ and $k = N$ are considered as special cases. First $k = 0$ is treated

$$\nabla_{x_0} \mathcal{L}(w, \lambda) = \nabla_{x_0} L(x_0, u_0) + \frac{\partial f}{\partial x_0}(x_0, u_0)^T \lambda_1 - \frac{\partial r}{\partial x_0}(x_0, x_N)^T \lambda_r = 0. \tag{13.12a}$$

Then the case for $k = 1, \ldots, N-1$ is treated

$$\nabla_{x_k} \mathcal{L}(w, \lambda) = \nabla_{x_k} L(x_k, u_k) - \lambda_k + \frac{\partial f}{\partial x_k}(x_k, u_k)^T \lambda_{k+1} = 0. \tag{13.12b}$$

Now the special case $k = N$

$$\nabla_{x_N} \mathcal{L}(w, \lambda) = \nabla_{x_N} E(x_N) - \lambda_N - \frac{\partial r}{\partial x_N}(x_0, x_N)^T \lambda_r = 0. \tag{13.12c}$$

The Lagrangian with respect to u is calculated, for $k = 0, \ldots, N-1$

$$\nabla_{u_k} \mathcal{L}(w, \lambda) = \nabla_{u_k} L(x_k, u_k) + \frac{\partial f}{\partial u_k}(x_k, u_k)^T \lambda_{k+1} = 0. \tag{13.12d}$$

The last two conditions are

$$x_{k+1} - f(x_k, u_k) = 0 \quad k = 0, \ldots, N-1 \tag{13.12e}$$
$$r(x_0, x_N) = 0 \tag{13.12f}$$

The equations (13.12a) till (13.12f) are the KKT-system of the OCP. There exist different approaches to solve this system. One method is to solve equations (13.12a) to (13.12f) directly, this is called the simultaneous approach. The other approach is to calculate all the states in (13.12e) by forward elemination. This is called the sequential approach and treated first.

13.3 Sequential Approach to Optimal Control

This method is also called "single shooting" or "reduced approach". The idea is to keep only x_0 and $U = [u_0^T, \ldots, u_{N-1}^T]^T$ as variables. The states $x_1, \ldots, x_N$ are eleminated recursively by

$$\begin{aligned} \bar{x}_0(x_0, U) &= x_0 && (13.13) \\ \bar{x}_{k+1}(x_0, U) &= f(\bar{x}_k(x_0, U), u_k) && (13.14) \end{aligned}$$

Then the optimal control problem is equivalent to a problem with less variables

$$\begin{aligned} \underset{x_0, U}{\text{minimize}} \quad & \sum_{k=0}^{N-1} L(\bar{x}_k(x_0, U), u_k) + E(\bar{x}_N(x_0, U)) && (13.15a) \\ \text{subject to} \quad & r(x_0, \bar{x}_N(x_0, U)) = 0 && (13.15b) \end{aligned}$$

Note that equation (13.12e) is implicitly satisfied. This is called the reduced optimal control problem. It can be solved by e.g. Newton type method (SQP if inequalities are present). If $r(x_0, x_N) = x_0 - \bar{x}_0$ one can also eliminate $x_0 \equiv \bar{x}_0$. The optimality conditions for this problem are found in the next subsection.

13.4 Backward Differentiation of Sequential Lagrangian

The Lagrangian function is given by

$$\bar{\mathcal{L}}(x_0, U, \lambda_r) = \sum_{k=0}^{N-1} L(\bar{x}_k(x_0, U), u_k) + E(\bar{x}_k(x_0, U)) - \lambda_r^T r(x_0, \bar{x}_N(x_0, U)) \qquad (13.16)$$

so the KKT conditions for the reduced optimal control problem are

$$\begin{aligned} \nabla_{x_0} \bar{\mathcal{L}}(x_0, U, \lambda_r) &= 0 && (13.17a) \\ \nabla_{u_k} \bar{\mathcal{L}}(x_0, U, \lambda_r) &= 0 \quad k = 0, \ldots, N-1 && (13.17b) \\ r(x_0, \bar{x}_N(x_0, U)) &= 0 && (13.17c) \end{aligned}$$

Usually derivatives are computed by finite differences, the I-Trick or forward automatic differention (AD). But here, backward automatic differentiation (AD) is more efficient. The result for backward AD to the equations (13.17a) to (13.17c) to get $\nabla_{x_0} \bar{\mathcal{L}}$ and $\nabla_{u_k} \bar{\mathcal{L}}$ is stated in Algorithm 11. Compare this algorithm with equations (13.12a) to (13.12d) where $\bar{\lambda}_k \equiv \lambda_k$.

We get a second interpretation to the sequential approach with backward AD: when solving (13.12a) to (13.12f) we eliminate all equations that can be eliminated by (13.12e), (13.12c) and (13.12b). Only the equations (13.12f), (13.12a) and (13.12d) remain. Backward automatic differentiation (AD) gives the gradient at a cost scaling linearly with N and forward differences with respect to $u_0, \ldots, u_{N-1}$, would grow with N^2.

The sequential approach and backward automatic differentiation (AD) leads to a small *dense* (Jacobians are dense matrices) nonlinear system in variables $(x_0, u_0, \ldots, u_{N-1}, \lambda_r)$. The next section tries to avoid the dense Jacobians.

Algorithm 11 Result of backward AD to KKT-ROCP

Inputs
x_0, u_0,...,u_{N-1}, λ_r
Outputs
r, $\nabla_{u_0}\mathcal{L}, \ldots, \nabla_{u_{N-1}}\mathcal{L}$ and $\nabla_{x_0}\mathcal{L}$

Set $k = 0$, execute forward sweep:
repeat
$x_{k+1} = f(x_k, u_k)$
$k = k + 1$
until $k = N - 1$
Get $r(x_0, x_N)$

Set $\lambda_N = \nabla E(x_N) - \frac{\partial r}{\partial x_N}(x_0, x_N)^T \lambda_r$
Set $k = N - 1$, execute backward sweep:
repeat
$\lambda_k = \nabla_{x_k} L(x_k, u_k) + \frac{\partial f}{\partial x_k}(x_k, u_k)^T \lambda_{k+1}$
$\nabla_{u_k}\mathcal{L} = \nabla_{u_k} L(x_k, u_k) + \frac{\partial f}{\partial u_k}(x_k, u_k)^T \lambda_{k+1}$
$k = k - 1$
until $k = 0$

Compute $\nabla_{x_0}\mathcal{L} = \lambda_0 - \frac{\partial r}{\partial x_0}(x_0, x_N)^T \lambda_r$

13.5 Simultaneous Optimal Control

This method is also called "multiple shooting" or "one shot optimization". The idea is to solve (13.12a) to (13.12f) directly by a sparsity exploiting Newton-type method. If we regard the original OCP, it is an NLP in variables $w = (x_0, u_0, x_1, u_1, \ldots, u_{N-1}, x_N)$ with multipliers $(\lambda_1, \ldots, \lambda_N, \lambda_r) = \lambda$. In the SQP method we get

$$w_{k+1} = w_k + \Delta w_k \tag{13.18}$$
$$\lambda_{k+1} = \lambda_k^{QP} \tag{13.19}$$

by solving

$$\underset{\Delta w}{\text{minimize}} \quad \nabla_w F(w_k)^T \Delta w + \frac{1}{2}\Delta w^T B_k \Delta w \tag{13.20a}$$
$$\text{subject to} \quad G(w) + \frac{\partial G}{\partial w}(w)\Delta w = 0 \tag{13.20b}$$

If we use

$$B_k = \nabla_w^2 \mathcal{L}(w_k, \lambda_k) \tag{13.21}$$

this QP is very structured and equivalent to

$$\begin{aligned} \underset{\Delta x_0, \Delta u_0, \ldots, \Delta x_N}{\text{minimize}} \quad & \frac{1}{2}\sum_{k=0}^{N-1} \begin{bmatrix}\Delta x_k \\ \Delta u_k\end{bmatrix}^T Q_k \begin{bmatrix}\Delta x_k \\ \Delta u_k\end{bmatrix} + \frac{1}{2}\Delta x_N^T Q_N \Delta x_N + \sum_{k=0}^{N} \begin{bmatrix}\Delta x_N \\ \Delta u_N\end{bmatrix}^T g_k + \Delta x_N^T g_N \\ \text{subject to} \quad & r(x_0, x_N) + \frac{\partial r(x_0, x_N)}{\partial x_0}\Delta x_0 + \frac{\partial r(x_0, x_N)}{\partial x_N}\Delta x_N = 0 \\ & x_{k+1} - f(x_k, u_k) + \Delta x_{k+1} - A_k \Delta x_k - B_k \Delta u_k = 0 \quad \text{for} \quad k = 0, \ldots, N-1, \end{aligned}$$

with

$$Q_k = \nabla^2_{(x_k, u_k)} \mathcal{L}, \tag{13.22}$$
$$Q_N = \nabla^2_{x_N} \mathcal{L}, \tag{13.23}$$
$$g_k = \nabla_{(x_k, u_k)} L(x_k, u_k), \tag{13.24}$$
$$g_N = \nabla_x E(x_N), \tag{13.25}$$
$$A_k = \frac{\partial f}{\partial x_k}(x_k, u_k), \quad k = 0, \ldots, N-1, \tag{13.26}$$
$$B_k = \frac{\partial f}{\partial u_k}(x_k, u_k), \quad k = 0, \ldots, N-1. \tag{13.27}$$

Note that for $k \neq m$

$$\frac{\partial}{\partial x_k}\frac{\partial}{\partial x_m}\mathcal{L} = 0 \tag{13.28a}$$
$$\frac{\partial}{\partial x_k}\frac{\partial}{\partial u_m}\mathcal{L} = 0 \tag{13.28b}$$
$$\frac{\partial^2}{\partial u_k \partial u_m}\mathcal{L} = 0 \tag{13.28c}$$

This QP leads to a very sparse linear system and can be solved at a cost linear with N. Simultaneous approaches can deal better with unstable systems $x_{k+1} = f(x_k, u_k)$.

Appendix A

Example Report on Student Optimization Projects

In this section a project report written by students of a previous year at the end of the exercise sessions is presented. It comes in the original form without corrections (which would still be applicable), but might serve as an example of how such a report might look like.

A.1 Optimal Trajectory Design for a Servo Pneumatic Traction System

by Thijs Dewilde and Dries Van Overbeke

A.1.1 Introduction

Servo Pneumatic Positioning The system consists of an electromechanical actuator, the valve, and a pneumatic actuator or cylinder. (Figure A.1)

An electrically controlled valve with 5 ports and 3 switch positions drives a double-acting pneumatic cylinder. A linear unit is formed by combining the cylinder, piston and slider. The presented valve blocks the mass flows in its center switch position. Also, the valve is proportional which means it can switch continuously between positions. For a desired direction of movement, the piston is preset by controlling the valve accordingly. The valve is able to regulate the air mass flow rate and thus controls the movement of the piston.

Model Equations and States We assume the mass flows $\dot{m}$ are proportional to the valve control input u: $\dot{m}_1 \sim u$ and $\dot{m}_2 \sim u$. The time-dependant (t) model states are the cylinder chamber pressures P_1 and P_2, the velocity v and the position s. We define the model state vector

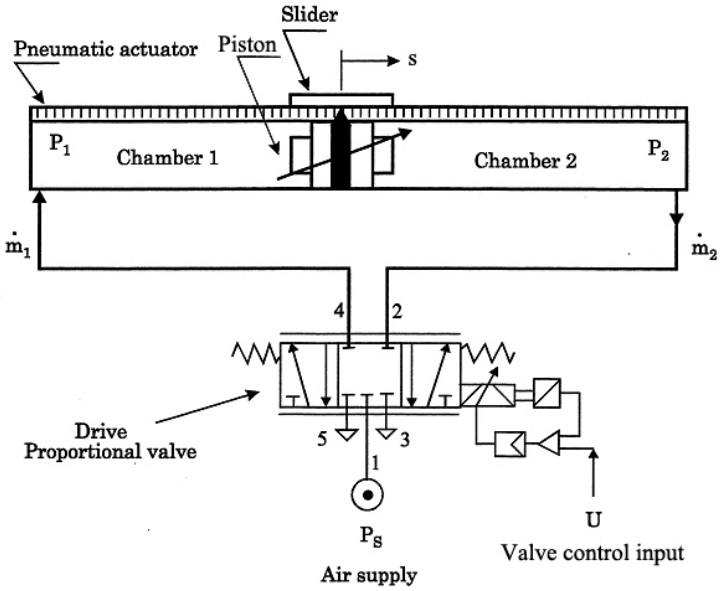


Figure A.1: Servo Pneumatic Traction System

x:

$$x = [P_1(t), P_2(t), v(t), s(t)]^T.$$

The volumes of the chambers can be computed by the following equations:

$$V_1 = V_t + A_c \cdot (s_{\text{ref}} + s),$$
$$V_2 = V_t + A_c \cdot (L - (s_{\text{ref}} + s)),$$

with
- s_{ref} : Reference position,
- s : Relative position of the piston,
- L : Cilinder stroke,
- D_c : Cilinder diameter,
- A_c : Cilinder chamber area,
- L_t : Tube length,
- D_t : Tube diameter,
- V_t : Tube volume.

Now consider an isothermic filling and venting process and differentiate the ideal gas law

$$p_i \cdot V_i = m_i \cdot R \cdot T_i,$$

with
- P_i : absolute pressure in chamber i,
- V_i : volume of chamber i, see equations $(A.1)$,
- m_i : mass of air in chamber i,
- R : specific gas constant for air $(287 \frac{J}{kg \cdot K})$,
- T_i : absolute temperature of the air in chamber i.

The following expressions for the pressure in the two chambers of the cylinder can be found:

$$\dot{p}_1 = \frac{R \cdot T \cdot \dot{m}_1 - Ac \cdot p_1 \cdot v}{V_1},$$

$$\dot{p}_2 = \frac{R \cdot T \cdot \dot{m}_2 + Ac \cdot p_2 \cdot v}{V_2},$$

with $\dot{m}_1$: mass flow of air to chamber 1,
$\dot{m}_2$: mass flow of air to chamber 2,
v : velocity of the piston.

After evaluating the differential pressure $p = p1 - p2$, the traction force on the piston can be calculated $p \cdot Ac$. Newton's second law of motion $F = m \cdot a$ and considering a viscous friction force $b \cdot v$, yields:

$$a = \frac{p \cdot Ac - b \cdot v}{m},$$

with m : mass of the piston and slider,
b : viscous friction coefficient.

Hereby the motion of the slider is entirely modelled, the velocity and position can be derivated by kinematics laws. So we have a dynamic system of the form:

$$\dot{x} = f(x, u, \tau).$$

A.1.2 Optimization Problem

Optimal Trajectory The optimal trajectory in this case is the fastest way to reach a position setpoint or in other words the appropriate control input for the proportional valve.

Parameters The control input signal is divided into m intervals, with $m \in \mathbb{N}$. The optimization parameters are the length of a control time interval τ and valve control value for each interval $u(m)$.

Formulation The total elapsed time for reaching a position setpoint or time horizon is minimized, The time horizon T can be regarded as a parameter in the differential equation by a time-transformation $T = m \cdot \tau$, so the objective function looks as follows:

$$J(T).$$

The equality constraint function ensures we hold a fixed position setpoint s_{end} at the end, by requiring $v_{end} = 0\frac{m}{s}$ and $a_{end} = 0\frac{m}{s^2}$. We summarize these equality constraints in a function g: $\mathbb{R}^3 \times R^m \to \mathbb{R}^2$:

$$g(x(T)) = 0.$$

Finally, the inequality constraint function limits the control value $-5 \leq u \leq 5$ and guarantees a positive time interval $T \geq 0$:

$$h(u(m), T) \leq 0.$$

We can formulate the problem in standard form:

$$\begin{aligned} \underset{x \in \mathbb{R}^4, u \in \mathbb{R}^m, T}{\text{minimize}} \quad & J(T) \\ \text{subject to} \quad & \dot{x} = f(x, u(m), \tau) \\ & s_{\text{start}} = 0 \\ & v_{\text{start}} = 0 \\ & p_{1,\text{start}} = 0 \\ & p_{2,\text{start}} = 0 \\ & g(x(T)) = 0 \\ & h(u(m), T) \leq 0 \end{aligned}$$

The model states are updated by a discrete function: $\dot{x} = f(x(k), u(m), \tau)$, To achieve better state updates the time interval is divided into a number h^{-1} of discretization steps.

$$f(x(k), u(m), \tau) = \begin{cases} p_{1,k+1} = p_{1,k} + h \cdot \tau \cdot \frac{R \cdot T \cdot \dot{m}_1 - Ac \cdot p_{1,k} \cdot v}{V_1} \\ p_{2,k+1} = p_{2,k} + h \cdot \tau \cdot \frac{R \cdot T \cdot \dot{m}_2 + Ac \cdot p_{2,k} \cdot v}{V_2} \\ v_{k+1} = v_k + h \cdot \tau \cdot a \\ s_{k+1} = s_k + h \cdot \tau \cdot v \end{cases}$$

for $k \in \{1, \ldots, n\}$. The following initial states are chosen:

$$\begin{aligned} p_{1,\text{start}} &= \text{atmosphere pressure } (101325Pa), \\ p_{2,\text{start}} &= \text{atmosphere pressure}, \\ v_{\text{start}} &= 0\tfrac{m}{s}, \\ s_{\text{start}} &= 0m. \end{aligned}$$

Numerical Solution We present a solution obtained by a sequential quadratic programming method (Figure A.2). The relative position setpoint is $200mm$ and we take 10 degrees of freedom (m) for the controls values, so the time horizon T is $10 \cdot \tau$.

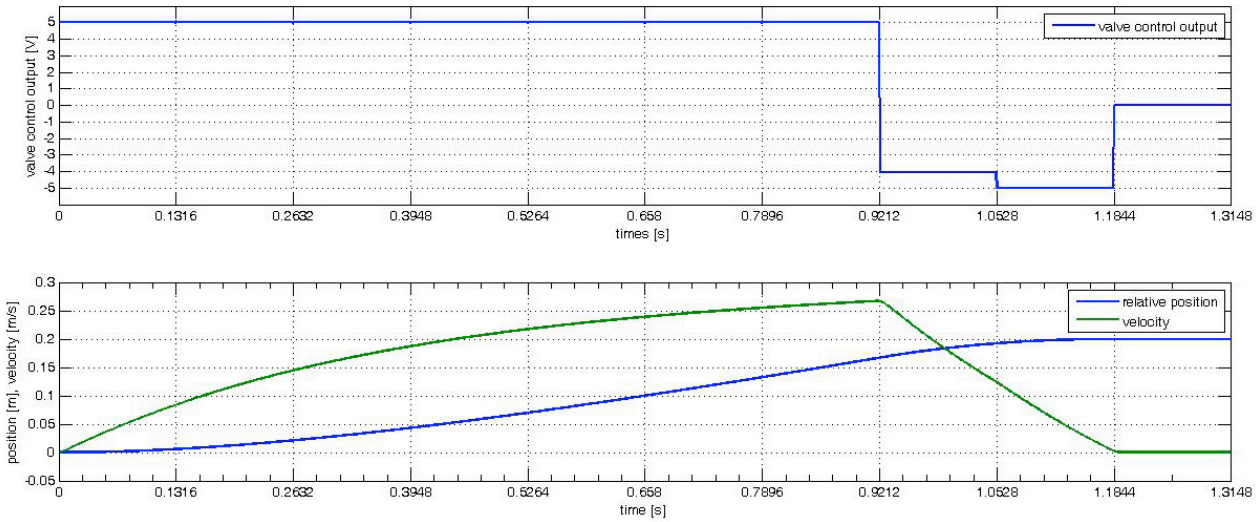


Figure A.2: Numerical Solution

In the plots we can see that the end condition constraints are satisfied and the presented control input is a "bang-bang" solution, with 2 degrees of freedom, namely the control value in between the limits and the control value for the last time interval. We can explain the first degree of freedom for reaching the setpoint position and the last control value to decrease the velocity and acceleration.

Appendix B

Exam Preparation

B.1 Study Guide

Important Chapters and Sections from the Book of Nocedal and Wright

Most, but not all, of the topics of the course are covered in the book by Nocedal and Wright. Particularly useful chapters and sections that are good reading to course participants are

- Appendix A.1 and A.2: all
- Chapter 1: all
- Chapter 2: all
- Chapter 3: Section 3.1, Algorithm 3.1, Section 3.3, Theorem 3.5
- Chapter 4: Algorithm 4.1
- Chapter 6: Formula (6.19)
- Chapter 8: all
- Chapter 10: Sections 10.1, 10.2, 10.3
- Chapter 12: all
- Chapter 16: Sections 16.1, 10.2
- Chapter 18: all
- Chapter 19: Section 19.1, 19.2

Important topics from the course that are not covered well in the book are the Constrained Gauss-Newton method and convex optimization.

Important Chapters and Sections from the Book of Boyd and Vandenberghe

Regarding convex optimization, all topics of the course are covered in the book by Boyd and Vandenberghe. Particularly useful chapters and sections that are good reading to course participants are

- Chapter 1: all
- Chapter 2: Sections 2.1, 2.2, 2.3
- Chapter 3: Sections 3.1, 3.2
- Chapter 4: Sections 4.1, 4.2, 4.3, 4.4, 4.6
- Chapter 5: Sections 5.1, 5.2

B.2 Rehearsal Questions

The following questions might help in rehearsing the contents of the course:

1. What is an optimization problem? Objective, degrees of freedom, constraints. Feasible set? Standard form of NLP.
2. Definition of global and local minimum.
3. Types of optimization problems: Linear / Quadratic programming (LP/QP), convex, smooth, integer, optimal control...
4. When is a function convex? Definition. If it is twice differentiable?
5. When is a set convex? Definition.
6. What is a "stationary" point?
7. How are gradient and Hessian of a scalar function f defined?
8. What are the first order necessary conditions for optimality (FONC) (unconstrained)?
9. What are the second order necessary conditions for optimality (SONC) (unconstrained)?
10. What are the second order sufficient conditions for optimality (SOSC) (unconstrained)?
11. Basic idea of iterative descent methods?
12. Definition of local convergence rates: q/r-linear, superlinear, quadratic?
13. What is a locally convergent, what a globally convergent algorithm? What does the term "globalisation" usually mean for optimizers ?

14. What is the Armijo condition? What is the reason that it is usually required in line search algorithms?

15. Why is satisfaction of Armijo condition alone not sufficient to guarantee convergence towards stationary points? Give a simple counterexample.

16. What is backtracking?

17. What is the local convergence rate of the steepest descent method?

18. What is Newton's method for solution of nonlinear equations $F(x) = 0$? How does it iterate, what is the motivation for it. How does it converge locally?

19. How works Newton's method for unconstrained optimization?

20. What are approximate Newton, or Newton type methods?

21. What is the idea behind Quasi-Newton methods?

22. What is the secant condition? How is it motivated?

23. What is the BFGS formula? Let s_k be the last step vector and y_k the (Lagrange-) gradient difference. Under which condition does it preserve positive definiteness?

24. Can any update formula satisfying the secant condition yield a positive definite Hessian if $y_k^T s_k < 0$?

25. What is a linear what a nonlinear least squares problem (unconstrained)?

26. How does the Gauss-Newton method iterate? When is it applicable?

27. When does the Gauss-Newton method perform well? What local convergence rate does it have?

28. Statistical motivation of least squares terms in estimation problems?

29. Difference between line search and trust region methods? Describe the basic idea of the trust region method for unconstrained optimization.

30. List three ways to compute derivatives with help of computers.

31. What errors occur when computing derivatives with finite differences? Do you know a rule of thumb of how large to choose the perturbation?

32. If a scalar function f can be evaluated up to accuracy TOL$= 10^{-6}$, how accurate can you compute its Hessian by twice applying finite forward differences?

33. What is the idea behind Automatic Differentiation (AD)? What is its main advantage?

34. Can AD be applied recursively in order to obtain higher order derivatives?

35. There are two ways of AD. Describe briefly. What are the advantages / disadvantages of the two, w.r.t. computation time, storage requirements?

36. Assume you have a simulation routine with $n = 10^6$ inputs and a scalar output that you want to minimize. If one simulation run takes one second, how long would it take to compute the gradient by finite differences (forward and central), how long by automatic differentiation (forward and backward mode)?

37. What is the standard form of a nonlinear program (NLP)? How is the lagrangian function defined? What is it useful for?

38. What is the constraint qualification (CQ), what is the linear indepencence constraint qualification (LICQ) at a point x?

39. What are the Karush-Kuhn-Tucker (KKT) conditions for optimality? What do they guarantee in terms of feasible descent directions of first order?

40. What are the first order necessary conditions for optimality (FONC) (constrained)?

41. What are the second order necessary conditions for optimality (SONC) (constrained)?

42. What are the second order sufficient conditions for optimality (SOSC) (constrained)?

43. What is the "active set"?

44. Give a standardform of a QP.

45. When is a QP convex?

46. What is the main idea of an active set strategy?

47. What is the main idea behind an SQP method (for inequality constrained problems)?

48. What is the L1-penalty function? Under which condition is it "exact", i.e. has the same local minima as the original NLP?

49. Under which condition does an SQP search direction deliver a descent direction for the L1-penalty function?

50. How works Newton's method for equality constrained optimization?

51. What convergence rate does an SQP method with Hessian updates (like BFGS) usually have?

52. What is a linear what a nonlinear least squares problem (constrained)?

53. What is the constrained Gauss-Newton method (CGN)? What convergence rate does it have, when does it converge well?

54. (Give an interpretation of the Lagrange multipliers as "shadow prices". How does this help in the optimizing practice?)

55. What is the basic idea of interior point methods? Compare them with active set methods. What are the advantages of each?

56. (What input format does a QP Solver like `quadprog` expect? Are you able to set up a simple QP and solve it using `quadprog`?)

57. (What input format does an NLP Solver like `fmincon` expect? Are you able to set up a simple NLP and solve it using `fmincon`?)

58. What input format does the general convex solver `cvx` expect? Are you able to set up a simple convex problem and solve it using `cvx`?

59. How is the Lagrangian function of a general NLP defined ?

60. How is the Lagrangian dual function of a general NLP defined ?

61. How is the Lagrangian dual problem of a general NLP defined ?

62. What is weak duality? To which problems does it apply?

63. What is strong duality? Under which sufficient conditions does it apply?

64. What is a semidefinite program (SDP)? Give a standardform.

65. How would you reformulate and solve the following eigenvalue optimization problem for a symmetric matrix?

$$\min_{x\in\mathbb{R}^n} \lambda_{\max}\left(A_0 + \sum_{i=1}^{n} A_i x_i\right)$$

with $A_0, A_1, \ldots, A_n \in \mathbb{R}^{m\times m}$ being symmetric matrices.

66. Are you able to set up a simple SDP and solve it using `cvx`?

B.3 Answers to Rehearsal Questions by Xu Gang

These answers to the rehearsal questions are made by Ph.D. student Xu Gang.

1. What is an optimization problem? Objective,degrees of freedom,constraints,feasible set? Standard form of NLP.

   An optimization problem consists of the following three ingredients.

   - An objective function,$f(x)$, that shall be maximized or minimized
   - decision variables,x,that can be chosen,and
   - constraint that shall be respected,e.g. of the form $g(x) = 0$ (equality constraints) or $h(x) \geq 0$ (inequality constraints)

   $$\min_{x\in\mathbb{R}^n} f(x) \qquad \text{subject to } \begin{cases} g(x) = 0 \\ h(x) \geq 0 \end{cases}$$

   here $f : \mathbb{R}^n \to \mathbb{R}, g : \mathbb{R}^n \to \mathbb{R}^p, h : \mathbb{R}^n \to \mathbb{R}^q$

   x is the vector of variables,also called unknown parameters;
   f is the **objective function**,a function of x that we want to minimize or maximize;

g, h is the vector of **constraints** that the unknowns must satisfy.This is a vector function of the variables x

feasible set is $\Omega := \{x \in \mathbb{R}^n \mid g(x) = 0,\ h(x) \geq 0\}$.

2. Definition of global and local minimum.

   The point $x \in \mathbb{R}^n$ is a **global minimizer:** if and only if $x^* \in \Omega$ and $\forall x \in \Omega :\ f(x) \geq f(x^*)$

   The point $x \in \mathbb{R}^n$ is a **strict global minimizer:** if and only if $x^* \in \Omega$ and $\forall x \in \Omega \backslash \{x^*\} : f(x) > f(x^*)$

   The point $x \in \mathbb{R}^n$ is a **local minimizer:** if and only if $x^* \in \Omega$ and there exists a neighborhood N of x^* (e.g. an open ball around x^*) so that $\forall x \in \Omega \cap N :\ f(x) \geq f(x^*)$

   The point $x \in \mathbb{R}^n$ is a **strict local minimizer:** if and only if $x^* \in \Omega$ and there exists a neighborhood N of x^* so that $\forall x \in \Omega \cap N \backslash \{x^*\} :\ f(x) > f(x^*)$

3. When do minimizers exist?

   Theorem(weierstrass): If $\Omega \subset \mathbb{R}^n$ is **compact**(i.e.,bounded and closed) and $f : \Omega \to \mathbb{R}$ is continuous then there exists a **global** minimizer of the optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to } x \in \Omega$$

4. Types of optimization problems:Linear/Quadratic programming(LP/QP), convex,smooth,integer,optimal control...

   LP:

$$\min_{x \in \mathbb{R}^n} c^T x \qquad \text{subject to } \begin{cases} Ax - b = 0 \\ Cx - d \geq 0 \end{cases}$$

   $c \in \mathbb{R}^n, A \in \mathbb{R}^{p \times n}, b \in \mathbb{R}^p, C \in \mathbb{R}^{q \times n}, d \in \mathbb{R}^q$

   QP:

$$\min_{x \in \mathbb{R}^n} c^T x + \frac{1}{2} x^T B x \qquad \text{subject to } \begin{cases} Ax - b = 0 \\ Cx - d \geq 0 \end{cases}$$

   $c \in \mathbb{R}^n, A \in \mathbb{R}^{p \times n}, b \in \mathbb{R}^p, C \in \mathbb{R}^{q \times n}, d \in \mathbb{R}^q$, and Hessian $B \in \mathbb{R}^{n \times n}$

   Convex QP: when Hessian matrix B is **positive semi-definite**(i.e., if $\forall z \in \mathbb{R}^n : z^T B z \geq 0$)

   Strictly Convex QP: when Hessian matrix B is **positive definite**(i.e., if $\forall z \in \mathbb{R}^n \backslash \{0\} : z^T B z > 0$)

   Convex optimization problem: feasible set Ω is convex and objective function $f : \Omega \to \mathbb{R}$ is convex.

   Theorem: for a convex problem,every local minimum is also a global one.

   Convex maximization problem: A maximization problem $\max_{x \in \mathbb{R}^n} f(x) \quad s.t. \quad x \in \Omega$ is called a "convex maximization problem"if Ω is convex and f **concave**. It is equivalent to

convex minimization problem $\min_{x\in\mathbb{R}^n} -f(x) \quad s.t. \quad x \in \Omega$

Practically convex NLP: If in the NLP formulation the objective function f is convex,the equalities g are **affine**,and the inequalities h_i are **concave** functions,then the NLP is a convex optimization problem.

$$\min_{x\in\mathbb{R}^n} f_0(x) \qquad \text{subject to} \qquad \begin{cases} Ax = b \\ f_i(x) \leq 0, \quad i = 1, \ldots, m \end{cases}$$

$f_0, \ldots f_m$ are convex.

Quadratically constrained quadratic program(QCQP): with $f_i(x) = d_i + c_i^T x + \frac{1}{2}x^T B_i x$ with $B_i \geq 0$ for $i = 0, 1 \ldots m$

$$\min_{x\in\mathbb{R}^n} c_0^T x + \frac{1}{2}x^T B_0 x \qquad \text{subject to} \qquad \begin{cases} Ax = b \\ d_i + c_i^T x + \frac{1}{2}x^T B_i x \leq 0, \quad i = 1, \ldots, m \end{cases}$$

By choosing $B_1 = \cdots = B_m = 0$ we would obtain a usual QP, and by also setting $B_0 = 0$ we would obtain an LP.

Semidefinite programming(SDP):

$$\min_{x\in\mathbb{R}^n} c^T x \qquad \text{subject to} \qquad \begin{cases} Ax - b = 0 \\ B_0 + \sum_{i=1}^{n} B_i x_i \geq 0 \end{cases}$$

All LPs,QPs,QCQPs can also be formulated as SDPs,besides several other convex problems. Semidefinite programming is avery powerfual tool in convex optimization.

Non-smooth(non-differentiable) optimization problem: If one or more of the problem functions f, g, h are not differentiable.

Mixed-integer programming(MIP):

$$\min_{\substack{x\in\mathbb{R}^n \\ z\in\mathbb{Z}^m}} f(x,z) \quad \text{subject to} \quad \begin{matrix} g(x,z) = 0 \\ h(x,z) \geq 0 \end{matrix}$$

5. When is a function convex?definition. If it is twice differentiable?

Convex function: A function $f : \Omega \to \mathbb{R}$ is **convex**,if Ω is convex and if $\forall x, y \in \Omega, t \in [0,1] : f(x + t(y - x)) \leq f(x) + t(f(y) - f(x))$(all secants are above graph).

$f(x) = |x|$ is convex but does not have a derivative at point 0.(no need twice differentiable)

Theorem(convexity for C^2 functions): Assume that $f : \Omega \to \mathbb{R}$ is **twice continuously differentiable** and Ω convex.Then holds that f is convex **if and only if** for all $x \in \Omega$ the Hessian is **positive semi-definite**,i.e.,

$$\forall x \in \Omega : \qquad \nabla^2 f(x) \geq 0$$

The following operations preserve convexity of functions:

(a) Affine input transformation: If $f : \Omega \to \mathbb{R}$ is convex,then also $\tilde{f}(x) = f(Ax + b)$ (with$A \in \mathbb{R}^{n\times m}$) is convex on the domain $\tilde{\Omega} = \{x \in \mathbb{R}^m | Ax + b \in \Omega\}$.

(b) Concatenation with a monotone convex function: If $f : \Omega \to \mathbb{R}$ is convex and $g : \mathbb{R} \to \mathbb{R}$ is convex and monotonely increasing,then the function $g \circ f : \Omega \to \mathbb{R}, x \mapsto g(f(x))$ is also convex.

(c) The supremum over a set of convex functions $f_i(x), i \in I$ is convex:$f(x) = \sup_{i\in I} f_i(x)$.this can be proven by noting that the epigraph of f is the intersection of the epigraphs of f_i.

6. When is a set convex? Definition

Convex set: A set $\Omega \subset \mathbb{R}^n$ is convex,if $\forall x, y \in \Omega, t \in [0, 1] : x + t(y - x) \in \Omega$(all connecting lines lie inside set).

Theorem(convexity of sublevel sets): The sublevel set $\{x \in \Omega | f(x) \leq c\}$ of a **convex function** $f : \Omega \to \mathbb{R}$ with respect to any constant $c \in \mathbb{R}$ is convex.

The following operations preserve convexity of sets:

(a) The intersection of finitely or infinitely many convex sets is convex

(b) Affine image: if Ω is convex, then for $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$ also the set $A\Omega + b = \{y \in \mathbb{R}^m | \exists x \in \Omega : y = Ax + b\}$ is convex

(c) Affine pre-image: if Ω is convex, then for $A \in \mathbb{R}^{n\times m}, b \in \mathbb{R}^n$ also the set $\{z \in \mathbb{R}^m | Az + b \in \Omega\}$ is convex.

7. What is a "stationary" point?

stationary point is an input to a function where the derivative is zero (equivalently, the gradient is zero): where the function "stops" increasing or decreasing (hence the name).

Critical point is more general: a critical point is either a stationary point or a point where the derivative is not defined.

Descent direction: A vector $p \in \mathbb{R}^n$ with $\nabla f(x)^T p < 0$ is called a descent direction at x.

8. how are gradient and Hessian of scalar function f defined?

The **Gradient** of f is defined to be the **vector field** whose components are the **partial derivatives** of f.

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$$

which points in the direction of the greatest rate of increase of the scalar field, and whose magnitude is the greatest rate of change.

A generalization of the gradient for functions on a **Euclidean space** which have values in another Euclidean space is the **Jacobian**. A further generalization for a function from one **Banach space** to another is the **Frchet derivative**.

Jacobian: Suppose $F : \mathbb{R}^n \to \mathbb{R}^m$ is a function from Euclidean n-space to Euclidean

m-space.the Jacobian Matrix J ,as follows,

$$J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Hessian matrix is the square matrix of **second-order partial derivatives** of a function; that is, it describes the local curvature of a function of many variables. Given the **real-valued(scalar)** function $f(x_1, x_2, \ldots, x_n)$.The Hessian matrix as follows,

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

If f is instead **vector-valued**, i.e $f = (f_1, f_2, \ldots, f_n)$,then the array of second partial derivatives is not a two-dimensional matrix, but rather a tensor of rank 3.

9. **Theorem(First-order optimality condition for convex problems):** regard the convex optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad s.t. \quad x \in \Omega$$

with continuously differentiable objective function f. A point $x^* \in \Omega$ is a **global** optimizer **if and only if**

$$\forall y \in \Omega : \quad \nabla f(x^*)^T (y - x^*) \geq 0$$

Corollary(unconstrained convex problems): regard the unconstrained problem

$$\min_{x \in \mathbb{R}^n} f(x)$$

with $f(x)$ convex.Then a **necessary and sufficient** condition for x^* to be a **global** optimizer is

$$\nabla f(x^*) = 0$$

10. What are the **first order necessary conditions** for optimality(FONC)(unconstrained)?

    If x^* is a **local** minimizer and f is continuously differentiable in an open neighborhood of x^*, then

$$\nabla f(x^*) = 0$$

11. What are the **second order necessary conditions** for optimality (SONC) (unconstrained)?

    If x^* is a **local** minimizer of f and $\nabla^2 f$ is continuous in an open neighborhood of x^* then

$$\nabla f(x^*) = 0 \quad and \ \nabla^2 f(x^*) \geq 0$$

12. What are the **second order sufficient conditions** for optimality (SOSC) (unconstrained)?

    suppose that $\nabla^2 f$ is continuous in an open neighborhood of x^* and that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is **positive definite**.Then x^* is a **strict local minimizer** of f.

    this is not necessary for a stationary point x^* to be a **strict local minimizer**.
    (e.g.,$f(x) = x^4$,for which $x^* = 0$ is a strict local minimizer with $\nabla^2 f(x^*) = 0$).

13. Basic idea of iterative descent methods?

    An iterative algorithm generates a sequence $\{x^0, x^1, x^2, \dots\}$ of so called "iterates" with $x^k \to 0$.

14. Definition of local convergence rates:Q/R-linear,superlinear,quadratic?

    let $\{x_k\}$ be a sequence in $\mathbb{R}^n$ that converges to x^*.
    Q-linear: if there is a constant $r \in (0, 1)$ such that

    $$\limsup_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq r$$

    (e.g., $x^k = \frac{1}{2^k}$ and $x^k = 0.99^k$)
    Q-superlinear:

    $$\limsup_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0$$

    (e.g.,$x^k = \frac{1}{k!}$)
    Q-quadratic:

    $$\limsup_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq M$$

    where M is a positive constant,not necessarily less than 1.
    (e.g.,$x^k = \frac{1}{2^{2^k}}$)

    R-convergence: If norm sequence $\|x^k - \bar{x}\|$ is upper bounded by source sequence,$y^k \to 0,\ y^k \in \mathbb{R}$ i.e.,$\|x^k - \bar{x}\| < y^k$ and if y^k is converging

    - Q-linearly then x^k is R-linearly
    - Q-superlinearly then x^k is R-superlinearly
    - Q-quadratically then x^k is R-quadratically

    Q=Quotient R=Root

15. What is locally convergent,what a globally convergent algorithm? what does the term "globalisation" usually mean for optimizers?

    an iterative method is called **locally convergent** if the successive approximations produced by the method are guaranteed to converge to a solution when the **initial approximation is already close enough to the solution**. Iterative methods for nonlinear equations and their systems, such as Newton's method are usually only locally convergent.

    An iterative method that converges for **an arbitrary initial approximation** is called globally convergent. Iterative methods for systems of linear equations are usually globally convergent.

16. What is the Armijo condition? What is the reason that it is usually required in line search algorithms?

    Armijo stipulates that t_k should give **sufficient decrease** in f:

    $$f(x_k + t_k p_k) \leq f(x_k) + \gamma t_k \nabla f(x_k)^T p_k$$

    with $\gamma \in (0, \frac{1}{2})$ the relaxation of the gradient. In practice γ is chosen quite small,say $\gamma = 0.1$ or even smaller.
    This condition however is not sufficient to ensure that the algorithm makes fast enough progress.

17. Why is satisfaction of Armijo condition alone not sufficient to guarantee convergence towards sationary points?give a simple counterexample.

    It is satisfied for all sufficiently small values of t_k, so Armijo condition is not enough by itself to ensure that the algorithm makes reasonable progress.

18. What is Backtracking?

    Backtraking chooses the step length by starting with $t = 1$ and checking it against Armijo's condition.If Armijo is not satisfied, t will be reduced by a factor $\beta \in (0, 1)$. In practice β is chosen to be not too small to derivate not too much,e.g.,$\beta = 0.8$

19. What is the local convergence rate of the steepest descent method?

    Take $B_k = \alpha_k \mathbb{I}$ and $p_k = -B_k^{-1} \nabla f(x_k) = -\frac{\nabla f(x_k)}{\alpha_k}$ This is the **negative gradient** with convergence rate **Q-linear**

20. What is Newton's method for solution of nonlinear equations $F(x) = 0$? How does it iterate,what is the motivation for it.How does it converge locally?

    $$\nabla f(x_k) = \frac{\Delta y}{\Delta x} = \frac{f(x_k) - 0}{x_k - x_{k+1}} \Rightarrow \quad x_{k+1} = x_k - \frac{f(x_k)}{\nabla f(x_k)}$$

    It converges locally Q-quadratically. **Theorem(convergence of Newton's method):** suppose $f \in \mathbb{C}$ and moreover,$\nabla^2 f(x)$ is a Lipschitz function in a neighborhood of x^*. x^* is a local minimum satisfying SOSC($\nabla f(x^*) = 0$ *and* $\nabla^2 f(x^*) > 0$. If x_0 is sufficiently close to x^*,then Newton iterates $x_0, x_1, \ldots$

    - converges to x^*
    - converges with **quadratic** rate
    - sequence of $\|\nabla f(x_k)\|$ converges to zero **quadratically**

21. How works Newton's method for unconstrained optimization?

    - work with line-search (line search Newton methods)
    - work with trust-region(trust-region Newton methods)
    - the above two can work with Conjugate Gradient methods

22. what are approximation Newton or Newton type methods?

    Any iteration of form $x_{k+1} = x_k - B_k^{-1}\nabla f(x_k)$ with B_k **invertible** is called "Newton type iteration for optimization"

    - $B_k = \nabla^2 f(x_k)$—Newton's method
    - $B_k \approx \nabla^2 f(x_k)$—approximate Newton

23. What is the idea behind Quasi-Newton methods?

    (a) Approximate Hessian B_{k+1} from knowledge of B_k and $\nabla f(x_k)$ and $\nabla f(x_{k+1})$,we get the following **secant condition**

    $$B_{k+1}(x_{k+1} - x_k) = \nabla f(x_{k-1}) - \nabla f(x_k)$$

    (b) change previous estimate B_k only slightly,require simultaneously $B_{k+1} \approx B_k$ and the secant condition.

24. What is the secant condition?How is it motivated?

    First-order Taylor:$\nabla f(x_{k+1}) \approx \nabla f(x_k) + \nabla^2 f(x_k)(x_{k+1} - x_k)$

25. What is BFGS formula?Let s_k be the last step vector and y_k the (Lagrange-) gradient difference. Under which condition does it preserve position definiteness?

    BFGS:

    $$B_{k+1} = B_k - \frac{B_k S S^T B_k}{S^T B_k S} + \frac{Y Y^T}{S^T Y}$$

    with $S = x_{k+1} - x_k$ and $Y = \nabla f(x_{k+1}) - \nabla f(x_k)$

    easily check:

    - B_{k+1} is symmetric
    - $B_{k+1}S_k = Y_k$
    - $B_{k+1} - B_k$ has Rank-2

    If B_k is **positive definite** and $Y_k^T S_k > 0$ Then is B_{k+1} well defined and **positive definite**.

26. Can any update formula satisfying the secant condition yield a positive definite Hessian if $Y_k^T S_k < 0$?

    Lemma: If $Y_k^T S_k \leq 0$ and B_{k+1} satisfies secant condition,Then B_{k+1} **cannot be positive** definite.

27. What is linear what a nonlinear least-squares problem(unconstrained)?

    In least-square problems,the objective function f has the following special form:

    $$f(x) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(x) = \frac{1}{2}\|\eta - M(x)\|_2^2$$

where each r_j is a smooth function from $\mathbb{R}^n$ *to* $\mathbb{R}$(mostly $m >> n$),we refer to each r_j as a **residual**.

linear least-squares problems

In a special case in which each function r_i is linear,the Jacobian J is constant,and we can write

$$f(x) = \frac{1}{2}\|Jx + r\|_2^2 \; or \; = \frac{1}{2}\|\eta - Jx\|_2^2$$

where $r = r(0)$,we also have

$$\nabla f(x) = J^T(Jx + r), \qquad \nabla^2 f(x) = J^T J$$

(note that the second term in $\nabla^2 f(x)$ in disappears,because $\nabla^2 r_i = 0$ for all i)

nonlinear least-square problem

$$\min_x f(x) \quad with \quad f(x) = \frac{1}{2}\|\eta - M(x)\|_2^2$$

28. How does the Gauss-Newton method iterate? When is it applicable?

$x_{k+1} = x_k + p_k^{GN}$ with $J_k^T J_k p_k^{GN} = -J_k^T r_k$

$$p_k^{GN} = -(J^T J)^{-1} J^T F = -J^+ F$$

with $\nabla^2 f = J^T J$, $\nabla f = J^T F$ and $J^+ = (J^T J)^{-1} J^T$ the **pseudo-inverse**(numerically more stable to comput J^+ directly,e.g.,QR-factorization).

It is only applicable to estimation problems because the methods linearizes nonlinear function inside L2-norm in fitting problems.
(remark: $J^T J$ is not always invertible.

29. When does the Gauss-Newton method perform well?What local convergence rate does it have?

It converges Q-linear to x^*.

30. Statistical motivation of least squares terms in estimation problems?

A least squares problem can be interpreted as finding x that "explains" noisy measurements "best".

Definition: A maximum-likelyhood estimate maximize the probability $P(n|x)$ of obtaining the (given) measurements if the parameter has value x.

assume $\eta_i = M_i(\bar{x}) + \varepsilon_i$ with $\bar{x}$ the "true" parameter,and ε_i Gaussian noise(with expectation value $\mathbb{E}(\varepsilon_i) = 0$, $\mathbb{E}(\varepsilon_i, \varepsilon_i) = \sigma^2$ and $\varepsilon_i, \varepsilon_j$ independent).

$$P(\eta|x) = \prod_{i=1}^{m} P(\eta_i|x) = \prod_{i=1}^{m} C \exp\left(\frac{-(\eta_i - M_i(x))^2}{2\sigma^2}\right)$$

$$\log P(\eta|x) = C + \sum_{i=1}^{m} -\frac{-(\eta_i - M_i(x))^2}{2\sigma^2}$$

The **argument maximizing**:

$$\arg\max_{x\in\mathbb{R}^n} P(\eta|x) = \arg\min_{x\in\mathbb{R}^n} \frac{1}{2}\|\eta - M(x)\|_2^2$$

31. Difference between line search and trust region methods? Describe the basic idea of the trust region method for unconstrained problem.

    They both generate steps with the help of quadratic model of the objective function,but they use this model in different ways,line search methods use it to generate a search direction,and then focus their efforts on finding a suitable step length α along this direction. Trust-region methods define a region around the current iterate with in which they trust the model to be an adequate representation of the objective,and then choose the step to be the approximate minimizer for the model in this trust region.

    Trust-region method:

    Iterate: $x_{k+1} = x_k + p_k$ where p_k solves

    $$\min_p M_k(p) \quad \text{subject to} \quad \|p\|_2 \leq \Delta_k$$

    can be used in the case of **indefinite Hessian**.

32. List three ways to compute derivatives with help of computers.

    - Symbolic differentiation
    - "imaginary trick" in matlab
      If $f : \mathbb{R}^n \to \mathbb{R}$ is analytic,then for $t = 10^{-100}$ we have

      $$\nabla f(x)^T p = \frac{Im(f(x+itp))}{t}$$

      can be calculated up to machine precision.
    - numerical differentiation(finite difference)
      easy and fast but inaccurate. $\frac{f(x+tp)-f(x)}{t} \approx \nabla f(x)^T p$
    - Automatic differentiation(forward and reverse)

33. What errors occur when computing derivatives with finite differences? Do you know a rule of thumb of how large to choose the perturbation?

    If we take t too small the derivative will suffer from **numerical noise(round-off error)**. On the other hand,if we took t too large the **linearization error** will be dominant.
    A good rule of thumb is to use $t = \sqrt{\varepsilon_{mach}}$, with ε_{mach} the machine precision(or the precision of f, if it is lower than the machine precision)
    The accuracy of this method is $\sqrt{\varepsilon_{mach}}$,which means in practice only half the digits are useful.Second order derivatives are therefore more difficult to accurately calculate.

34. If a scalar function f can be evaluated up to accuracy $TOL = 10^{-6}$,how accurate can you compute its Hessian by twice applying finite forward differences?

    choose $\varepsilon_{mach} = TOL = 10^{-6}$, so
    ∇f can achieve accuracy with 10^{-3}, so Hessian only with 0.1.

35. What is the idea behind Automatic Differentiation(AD)?What is its main advantage?

    Use **chain rule** and **differentiate** each ϕ_i separately.
    it can achieve accuracy up to machine precision.

36. Can AD be applied recursively in order to obtain higher order derivatives?

    AD can be generalized, in the natural way, to second order and higher derivatives. However, the arithmetic rules quickly grow very complicated, complexity will be quadratic in the highest derivative degree. Instead, truncated Taylor series arithmetic is used. This is possible because the Taylor summands in a Taylor series of a function are products of known coefficients and derivatives of the function. Computations of Hessians using AD has proven useful in some optimization contexts.

37. There are two ways of AD,Describe briefly.What are the advantages/disadvantages of the two,w.r.t. computation time,storage requirements?

    pre-AD algorithm:

    Input: $x_1, x_2, \cdots, x_n$
    Output: x_{n+m}

    for $i = n+1$ *to* $n+m$ **do**
      $x_i \leftarrow \phi_i(x_1, \cdots, x_{i-1})$
    end for

    Forward AD algorithm:

    Input: $\dot{x}_1, \dot{x}_2, \cdots, \dot{x}_n$
    Output: $\dot{x}_{n+m}$

    for $i = n+1$ *to* $n+m$ **do**
      $\dot{x}_i \leftarrow \sum_{j<n+i} \frac{\partial \phi_{n+i}}{\partial x_j} \dot{x}_j$
    end for

    $\text{cost}(\nabla f) \leq 2n \text{ cost}(f)$
    AD forward is slightly more expensive than FD,but is exact up to machine precision

    Reverse AD algorithm:

    Input: all partial derivatives $\frac{\partial \phi_i}{\partial x_i}$
    Output: $\bar{x}_1, \cdots, \bar{x}_n$

    $\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_{n+m-1} \leftarrow 0$
    $\bar{x}_{n+m}$
    for $i = n+m$ *down to* $n+1$ **do**
     for all $i < j$ **do**
      $\bar{x}_i \leftarrow \bar{x}_i + \bar{x}_j \frac{\partial \phi_j}{\partial x_i}$
     end for
    end for

    $\text{cost}(\nabla f) \leq 5 \text{ cost}(f)$,regardless of the dimension n!

The only disadvantage is that,you have to store all intermediate variables. may cause memory problem.

FD & Imaginary trick: $\text{cost}(\nabla f) = n + 1\ \text{cost}(f)$

38. assume you have a simulation routine with $n = 10^6$ inputs and a scalar output that you want to minimize. If one simulation run takes one second,how long would it take to compute the gradient by finite differences(forward and central),how long by automatic differentiation(forward and backward mode)

forward FD: $10^6 + 1$
central FD: $2 * 10^6 + 1$
forward AD: $2 * 10^6$
backward AD: 5

39. What is the standard form of NLP? how is the Lagrangian function defined? What is it useful for?

standard form of NLP:

$$\min_{x \in \mathbb{R}^n} f(x) \qquad \text{subject to } \begin{cases} g(x) = 0 \\ h(x) \geq 0 \end{cases}$$

here $f : \mathbb{R}^n \to \mathbb{R}, g : \mathbb{R}^n \to \mathbb{R}^p, h : \mathbb{R}^n \to \mathbb{R}^q$

Lagrangian function:

$$L(x, \lambda, \mu) = f(x) - \lambda^T g(x) - \mu^T h(x)$$

$\lambda \in \mathbb{R}^p$ and $\mu \in \mathbb{R}^q$ are "Lagrange multiplier" or "dual variables" we typically require the inequality multiplier $\mu \geq 0$,while the sign of the equality multiplier λ is arbitrary.

To formulate Lagrangian Dual problem.

40. What is the constraint qualification(CQ),what is the linear independence constraint qualification(LICQ) at a point x?

In order for a minimum point x^* be KKT, it should satisfy some regularity condition, the most used ones are listed below:
$LICQ \Rightarrow MFCQ \Rightarrow CPLD \Rightarrow QNCQ$, $LICQ \Rightarrow CRCQ \Rightarrow CPLD \Rightarrow QNCQ$(and the converse are not true),although MFCQ is not equivalent to CRCQ. In practice weaker constraint qualifications are preferred since they provide stronger optimality conditions.

LICQ:LICQ holds at $x^* \in \Omega$ if and only if all vector $\nabla g_i(x^*)$ for $i \in \{1, 2 \cdots, m\}$ and $\nabla h_i(x^*)$ for $i \in A(x^*)$ are linearly independent.

41. What are the KKT conditions for optimality? what do they guarantee in terms of feasible descent direction of first order?

Theorem(Fist-order necessary condition[KKT]): suppose that x^* is a local solution of NLP and that the LICQ holds at x^*.Then there is a Lagrange multiplier vector $\lambda^* \in$

$\mathbb{R}^m$ *and* $\mu \in \mathbb{R}^q$,such that the following condition are satisfied at (x^*, λ^*):

$$\begin{aligned} \nabla_x L(x^*, \lambda^*) = \nabla f(x^*) - \nabla g(x^*)\lambda - \nabla h(x^*)\mu &= 0 \\ g(x^*) &= 0 \\ h(x^*) &\geq 0 \\ \mu &\geq 0 \\ \mu_i h_i(x^*) &= 0 \quad i = 1, 2 \cdots, q \end{aligned}$$

42. What are the first order necessary conditions for optimality(FONC)(constrained)?

    KKT condition and variants:

    - If x^* is a local minimum of the NLP then:
      (a) $x^* \in \Omega$
      (b) for all tangents $p \in T_\Omega(x^*)$ holds:$\nabla f(x^*)^T p \geq 0$
    - If LICQ holds at x^* and x^* is a local minimizer of the NLP then:
      (a) $x^* \in \Omega$
      (b) $\forall p \in F(x^*)^T p \geq 0$
    - KKT condition

    Definition(tangent): $p \in \mathbb{R}^n$ is called a "tangent"to Ω at $x^* \in \Omega$ if there exists a smooth curve $\bar{x}(t) : [0, \varepsilon) \to \mathbb{R}^n$ with $\bar{x}(0) = x^*,\ \bar{x}(t) \in \Omega,\ \forall\ t \in [0, \varepsilon)$ and $\frac{d\bar{x}}{dt}(0) = p$.

    Definition(tangent cone): the "tangent cone" $T_\Omega(x^*)$ of Ω at x^* is the set of all tangent vectors at x^*.

    Definition(linearized feasible cone):
    $F(x^*) = \left\{g | \nabla g_i(x^*)^T p = 0, \quad i = 1, 2, \cdots, m\ \&\ \nabla h_i(x^*)^T p \geq 0, \quad i \in A(x^*)\right\}$ is called the "linearized feasible cone "at $x^* \in \Omega$.

    Definition(critical cone): Regard the KKT point (x^*, λ, μ).The critical cone $C(x^*, \mu)$ is the following set:
    $C(x^*, \mu) = \left\{p | \nabla g(x^*)^T p = 0,\ \nabla h_i(x^*)^T p = 0\ if\ i \in A_+(x^*, \mu),\ \nabla h_i(x^*)^T p \geq 0\ if\ i \in A_0(x^*, \mu)\right\}$

43. What are the second order necessary conditions for optimality(SONC)(constrained)?

    Regard x^* with LICQ. If x^* is local minimizer of the NLP,then:

    (a) $\exists \lambda, \mu$ so that KKT condition hold;
    (b) $\forall p \in C(x^*, \mu)$ holds that $p^T \nabla_x^2 L(x^*, \lambda, \mu) p \geq 0$

44. What are the second order sufficient conditions for optimality(SOSC)(constrained)?

    If x^* satisfies LICQ and

    (a) $\exists \lambda, \mu$ so that KKT condition hold;
    (b) $\forall p \in C(x^*, \mu),\ p \neq 0$ holds that $p^T \nabla_x^2 L(x^*, \lambda, \mu) p > 0$

    then x^* is a local minimizer.

45. what is the "active set"?

$$A(x) = \varepsilon \cup \{i \in I \ | c_i(x) = 0\}$$

Definition(active constraint): an **inequality** constraint $h_i(x) \geq 0$ is called "active" at $x^* \in \Omega$ if and only if $h_i(x^*) = 0$ and otherwise "inactive".

Definition(active set): The index set $A(x^*) \subset \{1, 2, \cdots, q\}$ of **active constraints** is called the "active set".

46. Give a standard form of a QP.

QP:

$$\min_{x \in \mathbb{R}^n} c^T x + \frac{1}{2} x^T B x \qquad \text{subject to } \begin{cases} Ax - b = 0 \\ Cx - d \geq 0 \end{cases}$$

$c \in \mathbb{R}^n, A \in \mathbb{R}^{p \times n}, b \in \mathbb{R}^p, C \in \mathbb{R}^{q \times n}, d \in \mathbb{R}^q$, and Hessian $B \in \mathbb{R}^{n \times n}$

Convex QP: when Hessian matrix B is **positive semi-definite**(i.e., if $\forall z \in \mathbb{R}^n : z^T B z \geq 0$)

Strictly Convex QP: when Hessian matrix B is **positive definite**(i.e., if $\forall z \in \mathbb{R}^n \setminus \{0\} : z^T B z > 0$)

47. what is a QP convex?

when $B \geq 0$

48. What is the main idea of an active set strategy?(for QP?)

49. What is the main idea behind an SQP method(for inequality constrained problems)?

Regard the NLP

$$\min_x f(x) \quad s.t. \quad h(x) \geq 0$$

SQP solve it in each iteration the QP

$$\min_p \nabla f(x)^T p + \frac{1}{2} p^T B_k p \quad s.t. \quad h(x_k) + \frac{\partial h}{\partial x}(x_k) p \geq 0$$

50. What is the L1-penalty function?under which condition is it "exact",e.e,has the same local minima as the original NLP?

A popular nonsmooth penalty function for the general nonlinear programming problem NLP is the $l1$ penalty function:

$$\phi_1(x; \mu) = f(x) + \mu \sum_{i \in \varepsilon} |c_i(x)| + \mu \sum_{i \in I} [c_i(x)]^-$$

where $[y]^- = \max\{0, -y\}$.Its name derives from the fact that the penalty term is μ times the $l1$ norm of the constraint violation. Note that $\phi_1(x; \mu)$ is not differentiable at some x,because of the presence of the absolute value and $[.]^-$ functions.

Theorem(exactness of $l1$ penalty function): Suppose that x^* is a strict local solution of the NLP at which the first-order necessary conditions are satisfied ,with Lagrange multipliers $\lambda_i^*, i \in \varepsilon \cup I$.Then x^* is a local minimizer of $\phi_1(x;\mu)$ for all $\mu > \mu^*$ where $\mu^* = \|\lambda^*\|_\infty = \max_{i\in\varepsilon\cup I} |\lambda_i^*|$. If,in addition,the second-order sufficient condition hold and $\mu > \mu^*$,then x^* is a strict local minimizer of $\phi_1(x;\mu)$.

Idea: use "merit function" to measure progree is both **objective** and **constraints**.

Definition($L_1 - merit$ function): is defined to be $T_1(x) = f(x) + \sigma\|g(x)\|_1$ with $\sigma > 0$.

51. Under which condition does an SQP search direction deliver a descent direction for the $l1$-penalty function?

If $B > 0$ and $\sigma \geq \|\tilde{\lambda}\|_\infty$ then p is a descent direction of T_1.

Definition(directional derivative): the "directional derivative of F at x in direction p" is $DF(x)[p] = \lim_{t\to 0, t>0} \frac{F(x+tp)-F(x)}{r}$.

Lemma: If p &$\tilde{\lambda}$ solve $\begin{bmatrix} \nabla f \\ g \end{bmatrix} + \begin{bmatrix} B & \frac{\partial g}{\partial x}^T \\ \frac{\partial g}{\partial x} & 0 \end{bmatrix} \begin{bmatrix} p \\ -\tilde{\lambda} \end{bmatrix} = 0$ then

$$\begin{aligned} DT_1(x)[p] &= \nabla f(x)^T p - \sigma\|g(x)\|_1 \\ DT_1(x)[p] &\leq -p^T B p - (\sigma - \|\tilde{\lambda}\|_\infty)\|g(x)\|_1 \end{aligned}$$

52. How works Newton's method for equality constrained optimization?

The idea is to apply Newton's method to solve the nonlinear KKT conditions

$$\begin{aligned} \nabla L(x,\lambda) &= 0 \\ g(x) &= 0 \end{aligned}$$

define

$$\begin{bmatrix} x \\ \lambda \end{bmatrix} = w \quad and \quad F(w) = \begin{bmatrix} \nabla L(x,\lambda) \\ g(x) \end{bmatrix}$$

so that the optimization is just a nonlinear root finding problem $F(w) = 0$,which can be solve by Newton's method.

$$F(w_k) + \frac{\partial F}{\partial w_k}(w - w_k) = 0$$

writeen in terms of gradients:

$$\begin{bmatrix} \nabla_x L \\ g \end{bmatrix} + \begin{bmatrix} \nabla_x^2 L & \nabla g \\ \nabla g^T & 0 \end{bmatrix} \begin{bmatrix} x - x_k \\ -(\lambda - \lambda_k) \end{bmatrix} = 0$$

53. What convergence rate does an SQP method with Hessian updates(like BFGS) usually have?

Newton-type **constrained optimization** converges

- quadratically if $B_k = \nabla^2 L(x_k, \lambda_k)$
- superlinearly if $B_k \to \nabla^2 L(x_k, \lambda_k)$(BFGS)

- linearly if $\|B_k - \nabla^2 L(x_k, \lambda_k)\|$ is not too big (Gauss-newton)

54. What is a linear what a nonlinear least squares problem(constrained)

$$\min_x f(x) \quad \text{subject to} \begin{cases} Ax - b = 0 \\ Ax - b \geq 0 \end{cases}$$

with

$$f(x) = \frac{1}{2}\|\eta - Jx\|_2^2$$

$$\min_x f(x) \quad \text{subject to} \begin{cases} g(x) = 0 \\ h(x) \geq 0 \end{cases}$$

with

$$f(x) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(x) = \frac{1}{2}\|\eta - M(x)\|_2^2$$

55. What is the constrained Gauss-Newton method(CGN)? What convergence rate does it have,when does it converge well?

Regard:

$$\min_x \frac{1}{2}\|F(x)\|_2^2 \qquad \text{subject to} \quad g(x) = 0$$

Linearize both F and g get approximation by:

$$\min_x \frac{1}{2}\|F(x_k) + J(x_k)(x - x_k)\|_2^2 \qquad \text{subject to} \quad g(x_k) + \nabla g(x_k)^T(x - x_k) = 0$$

This is a LS-QP which is **convex**.note that no multipliers $\lambda_{k+1} are needed$
KKT

$$\begin{aligned} J^T J(x - x_k) + J^T F - \nabla g \lambda(x - x_k) &= 0 \\ g + \nabla g^T &= 0 \end{aligned}$$

The constrained Gauss-Newton gives a Newton type iteration with $B_k = J^T J$, for LS

$$\nabla_x^2 L(x, \lambda) = J(x)^T J(x) + \sum F_i(x) \nabla^2 F_i(x) - \sum \lambda_i \nabla^2 g_i(x)$$

One can show that $\|\lambda\|$ gets small if $\|F\|$ is small. As in unconstrained case, CGN converges well if $\|F\| = 0$ (Q-linear).

56. Give an interpretation of the Lagrange multipliers as "shadow prices",How does this help in the optimizing practice?

Loosely, the shadow price is the change in the objective value of the optimal solution of an optimization problem obtained by relaxing the constraint by one unit.

More formally, the shadow price is the value of the Lagrange multiplier at the optimal solution, which means that it is the infinitesimal change in the objective function arising from an infinitesimal change in the constraint. This follows from the fact that at the optimal solution the gradient of the objective function is a linear combination of the constraint function gradients with the weights equal to the Lagrange multipliers. Each constraint in an optimization problem has a shadow price or dual variable.

57. What is the basic idea of interior point methods?compare them with active set methods.What are the advantage of each?

    The **Interior point method** is an alternative for the **active set method** for QPs or LPs and for **SQP method**.The previous methods have broblems with the **non-smoothness** in the KKT-condition(b,c,d)[$for\ i = 1, 2, \ldots, q$:

    (a) $\nabla f(x) - \sum_{i=1}^{q} \nabla h_i(x)\mu_i = 0$

    (b) $h_i(x) \geq 0$

    (c) $\mu_i \geq 0$

    (d) $\mu_i h_i(x) = 0$

    The Interior point method's idea is to replace b,c and d by a smooth condition(which is an approximation):$h_i(x)\mu_i = \tau$ with $\tau > 0$ but small. The KKT-conditions now become a smooth root finding problem:

    $$\begin{aligned} \nabla f(x) - \sum_{i=1}^{q} \nabla h_i(x)\mu_i &= 0 \\ h_i(x)\mu_i - \tau &= 0 \quad i = 1, 2, \ldots, q \end{aligned}$$

    These conditions are called the IP-KKT conditions and can be solved by Newtons methods and yields solutions $\bar{x}(\tau)$ and $\bar{\mu}(\tau)$.
    we can show that for $\tau \to 0$

    $$\begin{aligned} \bar{x}(\tau) &\to x^* \\ \bar{\mu}(\tau) &\to \mu^* \end{aligned}$$

58. What input format does a QP solver like quadprog expect?

59. What input format does an NLP solver like fmincon expect?

60. How is the Lagrangian function of a general NLP defined?

    $$L(x, \lambda, \mu) = f(x) - \lambda^T g(x) - \mu^T h(x)$$

    $\lambda \in \mathbb{R}^p$ and $\mu \in \mathbb{R}^q$ are "Lagrange multiplier" or "dual variables" we typically require the inequality multiplier $\mu \geq 0$,while the sign of the equality multiplier λ is arbitrary.

    Primal optimization problem: we denote the globally optimal value of the objective function subject to the constraints as "primal optimal value" p^*,i.e.,

    $$p^* = \left(\min_{x \in \mathbb{R}^n} f(x) \quad s.t. \quad g(x) = 0,\ h(x) \geq 0 \right)$$

    and we will denote this optimization problem as the "primal optimization problem".

    Lemma(lower bound property of Lagrangian): If $\tilde{x}$ is a feasible point and $\mu \geq 0$,then

    $$L(\tilde{x}, \lambda, \mu) \leq f(\tilde{x})$$

61. How is the Lagrangian dual function of a general NLP defined?

We define the so called "Lagrange dual function" as the **unconstrained infimum** of the Lagrangian over x, for fixed multipliers λ, μ.

$$q(\lambda, \mu) = \inf_{x \in \mathbb{R}^n} L(x, \lambda, \mu)$$

This function will often take the value $-\infty$, in which case we will say that the pair (λ, μ) is "**dual infeasible**" .

Lemma(lower bound property of Lagrange dual): If $\mu \geq 0$ then

$$q(\lambda, \mu) \leq p^*$$

Theorem(concavity of Lagrange dual): The function $q : \mathbb{R}^q \times \mathbb{R}^q \to \mathbb{R}$ is concave,even if the original NLP was not convex.

62. How is the Lagrangian dual problem of a general NLP defined?

the "**dual problem**" with "**dual optimal value**" d^* is defined as the **convex maximization problem**

$$d^* = \left(\max_{\lambda \in \mathbb{R}^p, \mu \in \mathbb{R}^q} q(\lambda, \mu) \quad s.t. \quad \mu \geq 0 \right)$$

the dual problem is *always* **convex**,even if the so called "primal problem" is not.

dual of an LP:

$$p^* = \min_{x \in \mathbb{R}^n} c^T x \quad \text{subject to} \quad \begin{array}{l} Ax - b = 0 \\ Cx - d \geq 0 \end{array}$$

$$d^* = \max_{\lambda \in \mathbb{R}^p, \mu \in \mathbb{R}^q} \begin{bmatrix} b \\ d \end{bmatrix} \begin{bmatrix} \lambda \\ \mu \end{bmatrix} \quad \text{subject to} \quad \begin{array}{rcl} c - A^T\lambda - C^T\mu & = & 0 \\ \mu & \geq & 0 \end{array}$$

dual of a strictly convex QP($B > 0$)

$$p^* = \min_{x \in \mathbb{R}^n} c^T x + \frac{1}{2} x^T B x \quad \text{subject to} \quad \begin{array}{l} Ax - b = 0 \\ Cx - d \geq 0 \end{array}$$

$$d^* = \max_{\lambda \in \mathbb{R}^p, \mu \in \mathbb{R}^q} -\frac{1}{2} c^T B^{-1} c + \begin{bmatrix} b + AB^{-1}c \\ d + CB^{-1}c \end{bmatrix}^T \begin{bmatrix} \lambda \\ \mu \end{bmatrix} - \frac{1}{2} \begin{bmatrix} \lambda \\ \mu \end{bmatrix}^T \begin{bmatrix} A \\ C \end{bmatrix} B^{-1} \begin{bmatrix} A \\ C \end{bmatrix}^T \begin{bmatrix} \lambda \\ \mu \end{bmatrix} \quad \text{subject to} \quad \mu \geq 0$$

63. What is weak duality? To which problems does it apply?

$$d^* \leq q^*$$

This holds for any arbitrary optimization problem,but does only unhold its full strength in convex optimization,where very often holds a strong version of duality.

64. What is strong duality?Under which sufficient conditions does it apply?

    Strong duality: If the primal optimization problem is **convex** and a technical constraint qualification(e.g, Slater's condition) holds,then primal and dual objective are equal to each other

    $$d^* = q^*$$

    Strong duality allows us to reformulate a convex optimization problem into its dual.

65. What is a semidefinite programin (SDP)?give a standardform.

    make use of linear matrix inequalities(LMI) in order to describe the feasible set($B_0 + \sum_{i=1}^{n} B_i x_i \geq 0$) where the matrices $B_0, \ldots, B_m$ are all in the vector space $\mathfrak{s}^k$ of symmetric matrices of a given dimension $\mathbb{R}^{k \times k}$.

    $$\min_{x \in \mathbb{R}^n} c^T x \qquad \text{subject to} \quad \begin{cases} Ax - b = 0 \\ B_0 + \sum_{i=1}^{n} B_i x_i \geq 0 \end{cases}$$

    All LPs,QPs,QCQPs can also be formulated as SDPs,besides several other convex problems. Semidefinite programming is avery powerfual tool in convex optimization.

66. How would you reformulate and solve the following eigenvalue optimization problem for a symmetric matrix?

    $$\min_{x \in \mathbb{R}^n} \lambda_{max} \left(A_0 + \sum_{i=1}^{n} A_i x_i \right)$$

    with $A_0, A_1, \ldots, A_n \in \mathbb{R}^{m \times m}$ being symmetric matrices.

    reformulated as SDP by adding a **slack variable** $s \in \mathbb{R}$,

    $$\min_{s \in \mathbb{R}, x \in \mathbb{R}^n} s \qquad \text{subject to} \quad \mathbb{I}_k s - \sum_{i=1}^{n} A_i x_i - A_0 \geq 0$$

67. Are you able to set up a simple SDP and solve it using CVX?

Bibliography

[1] S. Boyd and L. Vandenberghe. Convex Optimization. University Press, Cambridge, 2004.

[2] A.R. Conn, N. Gould, and P.L. Toint. Trust-Region Methods. MPS/SIAM Series on Optimization. SIAM, Philadelphia, USA, 2000.

[3] A. Griewank and A. Walther. Evaluating Derivatives. SIAM, 2 edition, 2008.

[4] J. Nocedal and S.J. Wright. Numerical Optimization. Springer Series in Operations Research and Financial Engineering. Springer, 2 edition, 2006.