LECTURE NOTES FOR THE COURSE SC 4091

Optimization in Systems and Control

Ton van den Boom and Bart De Schutter

August 2015

TU Delft
Artikelnummer 06917690001

Delft Center for Systems and Control
Delft University of Technology
Mekelweg 2, 2628 CD Delft, The Netherlands
`http://www.dcsc.tudelft.nl/`

Preface

These lecture notes are a continuously improved, extended, and rewritten version of the lecture notes "Optimization in Systems and Control" (edition October 1998) by T. van den Boom, B. De Schutter, M. Verhaegen, and V. Verdult. We want to acknowledge the contribution of the latter two authors to the original lecture notes, parts of which are still present in the current edition. We also want to thank them for the fruitful discussions we had with them on various topics discussed in these lecture notes.

Introduction to the Lecture Notes

The problem of optimization deals with how to do things in the best possible manner. The solution of such an optimization problem often forms a crucial step in making decisions in engineering problems. In systems and control engineering, there are many problems that are too complex to be solved analytically, and thus iterative optimization techniques have to be applied. Examples of these problems, where optimization is applied, are:

- Design of multi-criteria controllers
- Clustering in fuzzy modeling
- Trajectory planning of robots
- Scheduling in process industry
- Estimation of system parameters
- Design of predictive controllers with input-saturation

Some of the above engineering problems will be discussed in this course, some in other courses, such as: *Filtering and Identification* (SC4040), *Model Predictive Control* (SC4060), *Control Systems Lab* (SC4070), *Knowledge-Based Control Systems* (SC4080) and *Fuzzy Logic and Engineering Applications* (SC4150). Except for solving technical problems, optimization is a widely used approach in other fields, such as economics and finance.

When we decide to solve an engineering problem using optimization techniques, we have to deal with three subproblems:

1. **Formulation of the engineering problem into an optimization problem:**
   Sometimes we are not at all sure what is meant by “best”, “as accurately as possible”, “fastest” and “lowest”. The formulation step is therefore to choose some quantities, typically functions of several variables. Next we minimize one of the functions, possibly subject to one or more constraints, described by the remaining functions.

2. **Initialization of the optimization algorithm:**
   When the engineering problem is put into a mathematical framework, properties of the optimization problems can be studied to decide which algorithm will be able to yield the best results, using an acceptable amount of computational power. The initialization deals with the choice of algorithms and with decisions about initial values for the parameters and the measure for desired accuracy of the solution.

3. **The optimization procedure itself:**
   There are various optimization techniques making use of specific properties of the functions, chosen in the formulation step. The algorithms to find the optimal solution are usually implemented in a software package or library. In this course the *MATLAB Optimization Toolbox* is used.

These lecture notes consist of two parts:

Part 1: Optimization Techniques:
In the first part the mathematical framework of optimization is described and a division is made into different classes of problems. For each class, specific optimization algorithms are available with specific properties concerning reliability, robustness, convergence speed and computational effort. We will also discuss how the MATLAB Optimization Toolbox can be used to solve optimization problems.

Part 2: Formulating the Controller Design Problem as an Optimization Problem:
In the second part of the lecture notes the formulation of an engineering problem into an optimization problem is illustrated by the design of a controller with multiple criteria. We also consider implementation issues (including appropriate initialization) and we illustrate the control design with an extensive worked example.

Notation

Here we list some of the symbols and acronyms that occur frequently in these lecture notes and with which the reader might not be familiar. The numbers in the last column refer to the page on which the symbol or concept in question is defined.

List of symbols

Sets

$\emptyset$	the empty set
$\mathbb{N}$	set of the nonnegative integers: $\mathbb{N} = \{0, 1, 2, \ldots\}$
$\mathbb{N}_0$	set of the positive integers: $\mathbb{N}_0 = \mathbb{N} \setminus \{0\}$
$\mathbb{Z}$	set of the integers: $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$\mathbb{R}$	set of the real numbers
$\mathbb{R}^+$	set of the nonnegative real numbers

Functions

$\operatorname{dom} f$	domain of definition of the function f
$\log x$	natural logarithm of x (i.e., the logarithm with base e)
$\mathscr{O}(f)$	any real function g such that $\limsup_{x \to \infty} \dfrac{\lvert g(x) \rvert}{f(x)}$ is finite

Matrices and vectors

$\mathbb{R}^{m \times n}$	set of the m by n matrices with real entries
$\mathbb{R}^n$	set of the real column vectors with n components: $\mathbb{R}^n = \mathbb{R}^{n \times 1}$
A^T	transpose of the matrix A
I	identity matrix
O	zero matrix
a_i	ith component of the vector a
a_{ij}, $(A)_{ij}$	entry of the matrix A on the ith row and the jth column
$A_{i,\bullet}$	ith row of the matrix A
$A_{\bullet,j}$	jth column of the matrix A

We use $\square$ to indicate the end of a proof or an example, and $\diamond$ to indicate the end of a remark.

Acronyms

LP	Linear Programming	20
LTI	Linear Time-Invariant	131
MIMO	Multiple-Input Multiple-Output	131
QP	Quadratic Programming	27
RMS	Root Mean Square	160
SISO	Single-Input Single-Output	131
SVD	Singular Value Decomposition	187

Contents

Part I

Optimization Techniques

Chapter 1

Introduction

1.1 Mathematical framework

In this section we present the mathematical framework and we discuss two classes of optimization problems.
Any properly defined optimization problem consists of four separate characteristics, namely:

- f or $f(x)$: the objective function or criterion that expresses the intention or goal.
- x : the parameter vector that can be used to optimize the objective function f.
- $h(x) = 0$: equality constraints that restrict the solution to a certain subset of the parameter space.
- $g(x) \leqslant 0$: inequality constraints that restrict the solution to a certain allowed region of the parameter space.

Some typical objective functions in systems and control are: difference between the output of a system and a reference signal, energy of the input signal, energy of the output signal, time to reach a given state, ...The parameter vector could contain the parameters of a PID controller, set points for an industrial process controller, design parameters of the system such as weights, lengths, diameters, etc. Some examples of constraints are

- On a commercial airplane the vertical acceleration should be less than a certain value for passenger comfort.
- In an audio amplifier the power of noise signals at the output must be sufficiently small for high fidelity.
- In papermaking the moisture content must be kept between prescribed values.

In an optimization problem the objective function f and the parameter vector x are always present. However, the elements with h and g can be omitted if no constraints are imposed on the values of the parameters. We say that a point x of the parameter space is *feasible* if it satisfies both $h(x) = 0$ and $g(x) \leqslant 0$; otherwise we say that the point is *infeasible*.
Depending on the presence or absence of the constraints we can distinguish two kinds of optimization problems:

Unconstrained optimization problem:

The unconstrained optimization problem is defined as a search for the minimum of the objective function f:

$$f(x^*) = \min_x f(x)$$

where x^* is

$$x^* = \arg\min_x f(x)$$

Constrained optimization problem:

The constrained optimization problem is defined as a search for the minimum of the objective function f:

$$f(x^*) = \min_x f(x) \tag{1.1}$$

$$\text{subject to } h(x) = 0 \text{ and } g(x) \leqslant 0 \ . \tag{1.2}$$

It has to be noted that a modification of the objective function or the constraints might yield another optimal solution.

Note that there is no fundamental difference between minimization and maximization problems since

$$\max_x f(x) = -\min_x \left(-f(x) \right) \ .$$

Therefore, we will only consider minimization problems in these lecture notes.

The selection of the objective function is, in general, a compromise between the desire to have an appropriate measure to express the desired goal (fitting measurements, good control behavior) and the potential of optimization techniques to locate the optimal solution.

1.2 Unimodality and convexity

In the previous section the mathematical framework was unfolded. Besides having an efficient method to find a solution of the optimization problem, it is necessary to ensure that the located minimum really reflects the optimal solution of the problem. The guarantee to be able to find the optimal solution depends, amongst other things, on the properties of the functions f, g and h. In this section we consider the concept of unimodality and (quasi)convexity.

Definition 1.1 (Convex set)
A set $\mathscr{C}$ in $\mathbb{R}^n$ is convex if for each pair $x, y \in \mathscr{C}$ and for all $\lambda \in [0,1]$ the next property holds:

$$(1-\lambda)x + \lambda y \in \mathscr{C} \ .$$

The definition implies that a set is convex if the line segment joining any two points in the set lies entirely within the set as given in Figure 1.1.

Example 1.2 A ball, an ellipsoid, a cube, a hyper-cube, a plane, a line, a half-space, a triangle, and the space $\mathbb{R}^n$ are all convex sets. The set $\{ (x,y) \mid xy \geqslant 0 \}$ and the set $\mathbb{R}^n \setminus \{0\}$ are not convex. □

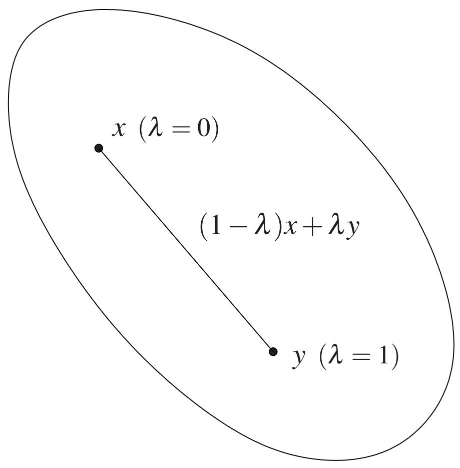


Figure 1.1: Convex set.

Definition 1.3 (Unimodal function)
A function f is unimodal if
(a) The domain $\mathrm{dom}(f)$ *is a convex set.*
(b) There exists an optimal solution $x^* \in \mathrm{dom}(f)$ such that

$$f(x^*) \leqslant f(x) \ \textit{for all} \ x \in \mathrm{dom}(f)$$

(c) For all $x_0 \in \mathrm{dom}(f)$, there exists a trajectory $x(\lambda) \in \mathrm{dom}(f)$, with $x(0) = x_0$ and $x(1) = x^*$ such that

$$f\Big(x(\lambda)\Big) \leqslant f(x_0) \ \textit{ for all } \ \lambda \in [0,1] \ .$$

The definition implies that there can be no local minima that are not global. From each point x_0 there exists a non-increasing trajectory to the optimum x^*.

Definition 1.4 (Quasiconvex function)
A function f is quasiconvex if
(a) The domain $\mathrm{dom}(f)$ *is a convex set.*
(b) The next inequality holds for all $x, y \in \mathrm{dom}(f)$ and $0 \leqslant \lambda \leqslant 1$:

$$f\Big((1-\lambda)x+\lambda y\Big) \leqslant \max\left(f(x), f(y)\right) \ .$$

The definition implies that a function is quasiconvex if the function curve between two points lies entirely under or on the maximum function value of these two points. Another characterization is given by the following lemma:

Lemma 1.5 *A function f is quasiconvex if and only if it has the following property: The set given by*

$$\mathscr{L}(\alpha) = \{\, x \in \mathrm{dom}(f) : \ f(x) \leqslant \alpha \,\}$$

is convex for every $\alpha \in \mathbb{R}$.

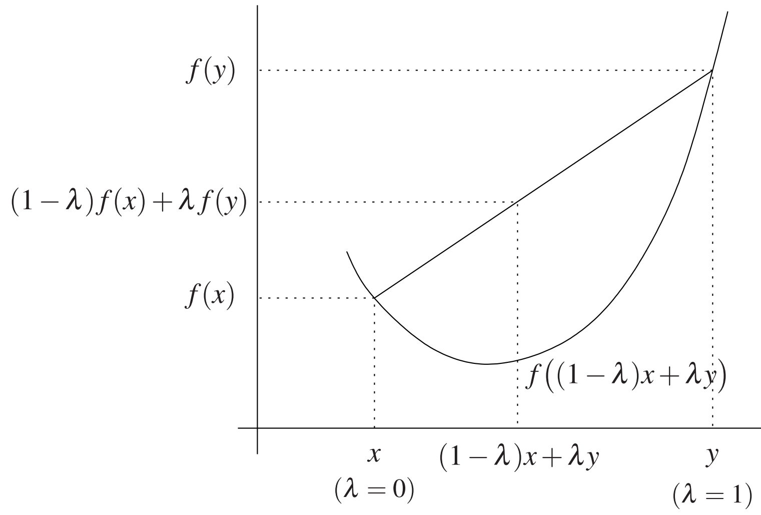


Figure 1.2: Convex function.

Definition 1.6 (Convex function)
A function f is convex if
(a) The domain $\mathrm{dom}(f)$ *is a convex set.*
(b) The next inequality holds for all $x, y \in \mathrm{dom}(f)$ and $0 \leqslant \lambda \leqslant 1$:

$$f\Big((1-\lambda)x+\lambda y\Big) \leqslant (1-\lambda)f(x)+\lambda f(y) \ .$$

The definition implies that a function is convex if the line segment joining any two points of the function curve lies above or on the function curve, as illustrated in Figure 1.2. More properties of convex functions are given in Section 6.1.

The definitions of unimodal, quasiconvex, and convex functions imply that a convex function is always quasiconvex and unimodal. A quasiconvex function is always unimodal, but not necessarily convex. A unimodal function is not necessarily convex or quasiconvex. Finally, in the one-dimensional case ($x \in \mathbb{R}$) unimodality and quasiconvexity are equivalent.

Example 1.7 The following functions are all convex:

$$\begin{aligned} &f(x) = |x|, \quad x \in \mathbb{R} \\ &f(x) = x^T x, \quad x \in \mathbb{R}^n \\ &f(x) = \max(\, e^{-x}, 1+x/4, x-2\,), \quad x \in \mathbb{R} \ . \end{aligned}$$ □

Example 1.8 The following function, known as the inverted Mexican hat function, is quasiconvex, but *not* convex:

$$f(x) = \frac{x^T x}{1+x^T x}\,, \, x \in \mathbb{R}^2$$

The three-dimensional plot of this function is given in Figure 1.3. From the contour plot of this function (Figure 1.4) it is clear that the interior of each of the contour lines (and thus also the corresponding level set[1] $\mathscr{L}(\alpha)$ with $\alpha \in \mathbb{R}$) is convex, and so the inverted Mexican hat function is quasiconvex. □

[1]Note that in fact this level set is the union of the interior of the contour line and the contour line itself.

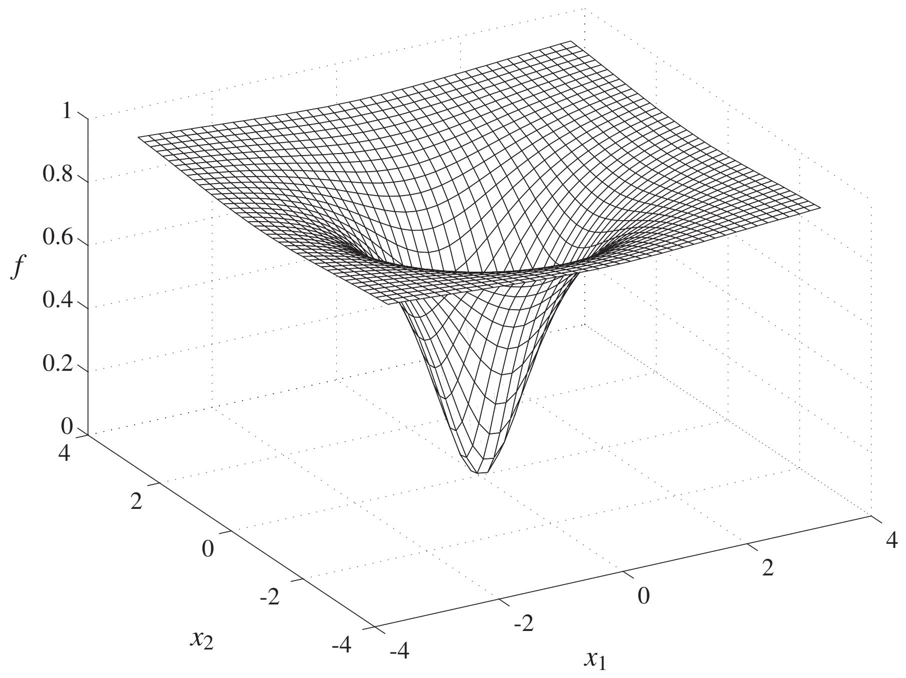


Figure 1.3: Inverted Mexican hat function.

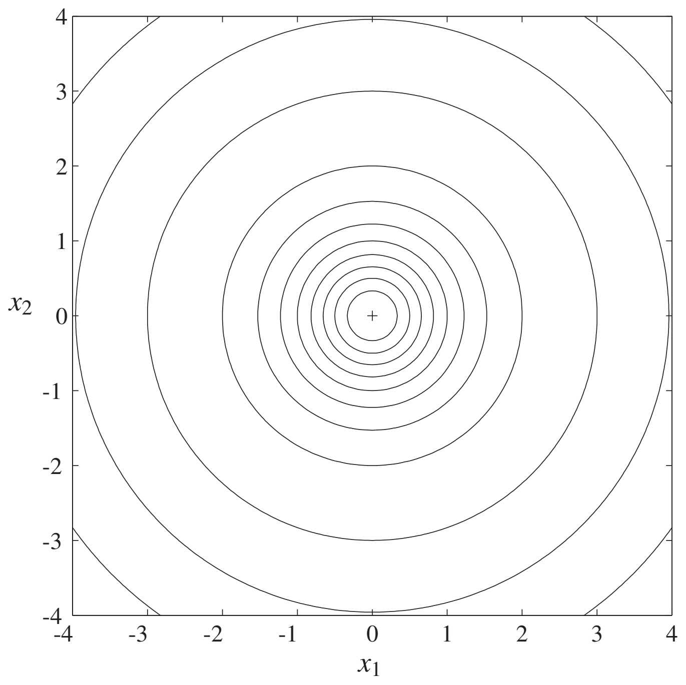


Figure 1.4: Contour plot of the inverted Mexican hat function. The position of the minimum is indicated by the $+$-mark.

Example 1.9 The following function, known as Rosenbrock's function, is unimodal, but *not* quasiconvex:

$$f(x_1,x_2) = 100\,(x_2 - x_1^2)^2 + (1-x_1)^2$$

The three-dimensional plot of Rosenbrock's function is given in Figure 1.5. From the contour plot of the function (Figure 1.6) it is clear that the interior of the contour lines are not convex, and so Rosenbrock's function is not quasiconvex. However, from each point $(x_1,x_2) \in \mathbb{R}^2$ there is a decreasing trajectory to the optimum $(1,1)$ and thus the function is unimodal. $\square$

Finally, we give some important definitions for gradient, Hessian, Jacobian and subgradient functions:
The gradient of the objective function f is defined by:

$$\nabla_x f(x) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} & \dfrac{\partial f}{\partial x_2} & \cdots & \dfrac{\partial f}{\partial x_n} \end{bmatrix}^T$$

If it is clear from the context that the partial derivatives have to be taken with respect to the components of x, then we drop the subscript x and we denote the gradient of f by ∇f.
Let us now briefly discuss the graphical interpretation of the gradient. Recall that the directional derivative of the function $f : \mathbb{R}^n \to \mathbb{R}$ in x_0 in the direction of the unit vector β is given by

$$D_\beta f(x_0) = \nabla^T f(x_0) \cdot \beta = \|\nabla f(x_0)\|_2 \cos\theta \ ,$$

where θ is the angle between $\nabla f(x_0)$ and β (see Figure 1.7). As a consequence, we have:

- Since $D_\beta f(x_0)$ is maximal if $\nabla f(x_0)$ and β are parallel, the function values exhibit the largest increase in the direction of $\nabla f(x_0)$ for points in the immediate neighborhood of x_0. Therefore, $-\nabla f(x_0)$ is called the *steepest descent direction.*

- Since $D_\beta f(x_0)$ is equal to 0 — i.e., the function values f do not change – if $\nabla f(x_0)$ and β are perpendicular, the gradient $\nabla f(x_0)$ is perpendicular to the contour line through x_0.

The Hessian of the objective function f is given by:

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

The Jacobian of the equality constraint function $h : \mathbb{R}^n \to \mathbb{R}^m$ is given by:

$$\nabla h(x) = \begin{bmatrix} \dfrac{\partial h_1}{\partial x_1} & \dfrac{\partial h_2}{\partial x_1} & \cdots & \dfrac{\partial h_m}{\partial x_1} \\ \dfrac{\partial h_1}{\partial x_2} & \dfrac{\partial h_2}{\partial x_2} & \cdots & \dfrac{\partial h_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial h_1}{\partial x_n} & \dfrac{\partial h_2}{\partial x_n} & \cdots & \dfrac{\partial h_m}{\partial x_n} \end{bmatrix}$$

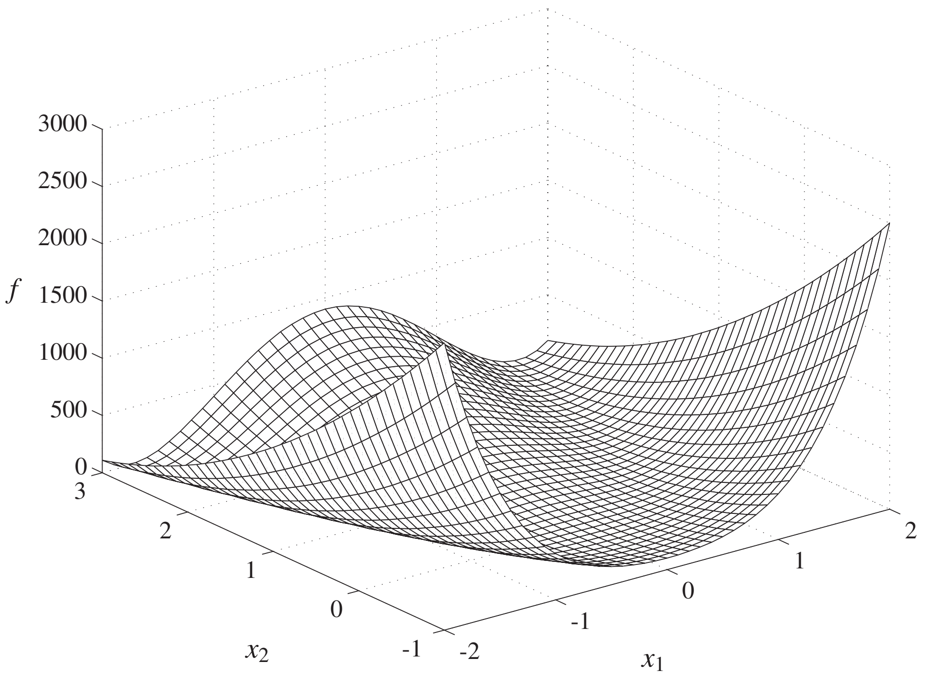


Figure 1.5: Rosenbrock's function.

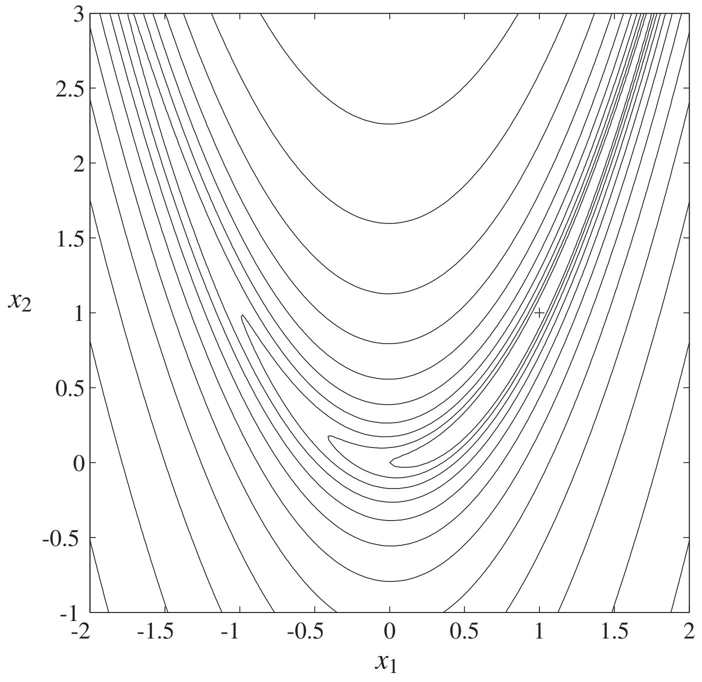


Figure 1.6: Contour plot of Rosenbrock's function. The position of the minimum is indicated by the +-mark.

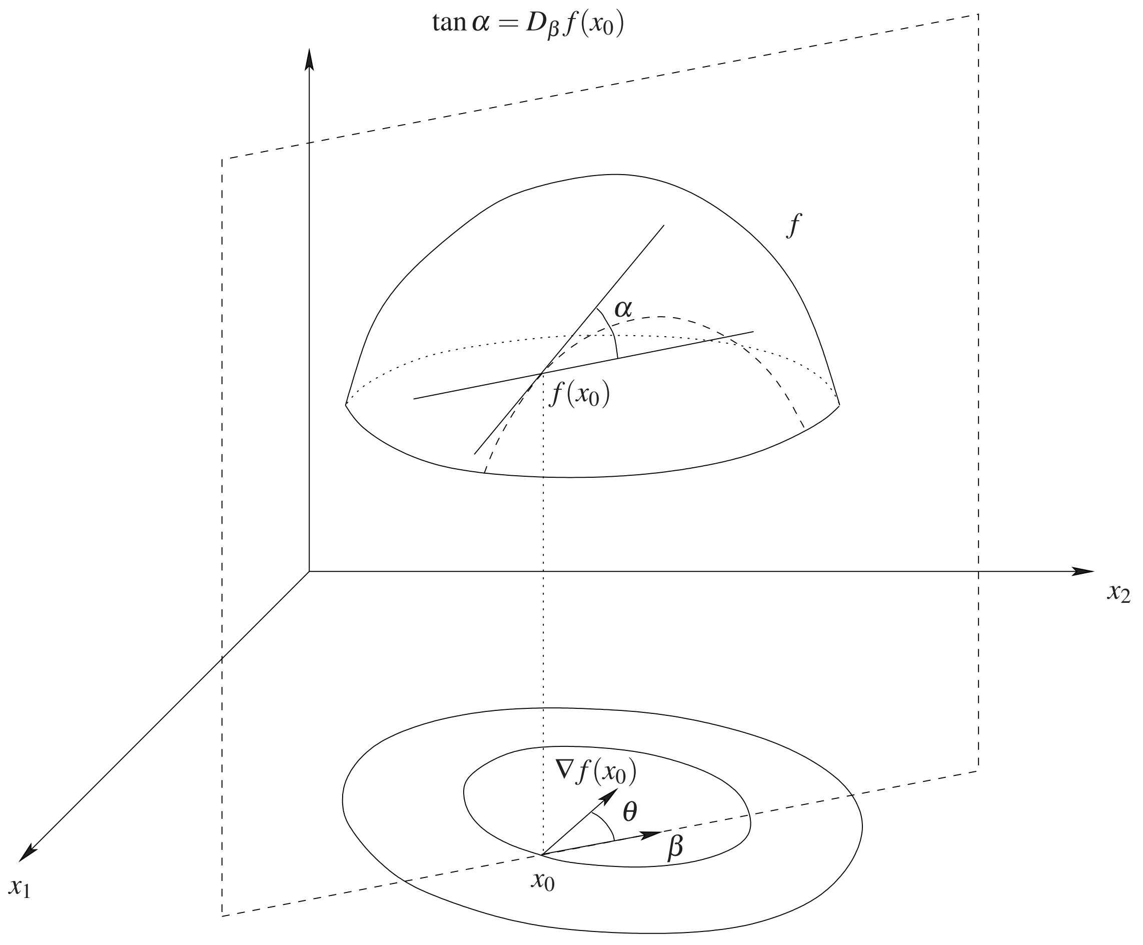


Figure 1.7: The graphical representation of the gradient $\nabla f(x_0)$ and the directional derivative $D_\beta f(x_0)$ of the function f.

The Jacobian of the inequality constraint function g is defined in a similar way. Note that for a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ the Jacobian coincides with the gradient, which motivates why the same symbol (∇) is used to denote the gradient and the Jacobian.

Finally, we consider the subgradient of a convex function:

Definition 1.10 (Subgradient)
Let f be a convex function. The function ∇f is called a subgradient of f if

$$f(x) \geqslant f(y) + \left(\nabla f(y)\right)^T (x-y) \qquad \textit{for all } x,y \in \operatorname{dom}(f).$$

For convex functions that are continuously differentiable, the subgradient is equal to the gradient. That is why we use the same symbol to denote the gradient and the subgradient.

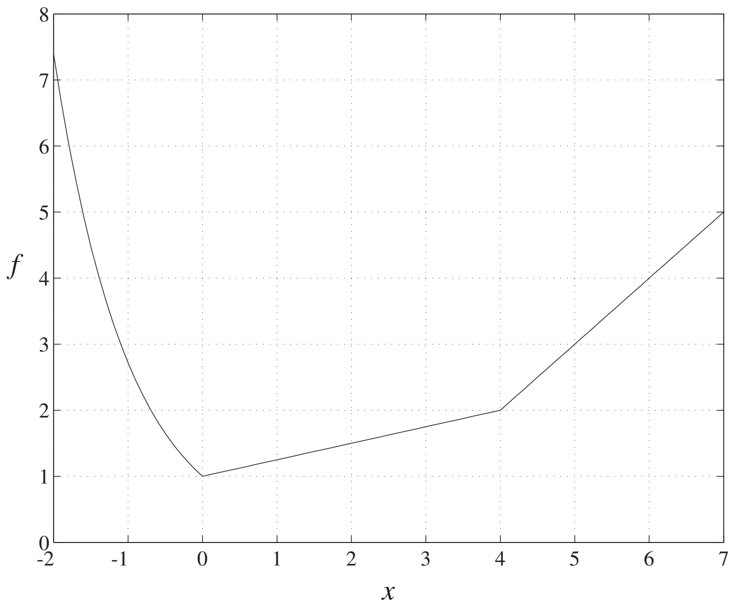


Figure 1.8: The graph of the function f defined by $f(x) = \max(\, e^{-x}, 1+x/4, x-2\,)$ with $x \in \mathbb{R}$.

Example 1.11 The convex functions, mentioned in Example 1.7, have the following subgradients:

$$\begin{aligned} f(x) = |x|,\ x \in \mathbb{R} \ &:\ \nabla f(x) = \operatorname{sign}(x) \\ f(x) = x^T x,\ x \in \mathbb{R}^n \ &:\ \nabla f(x) = 2x \\ f(x) = \max(\, e^{-x}, 1+x/4, x-2\,),\ x \in \mathbb{R} \ &:\ \nabla f(x) = \begin{cases} -e^{-x}, & x \leqslant 0 \\ 0.25, & 0 < x \leqslant 4 \\ 1, & x > 4 \end{cases} \end{aligned}$$

Note that in general the subgradient is not unique in the points in which the gradient of f is discontinuous or not defined. For the subgradient of the third function of this example in the points 0 and 4 we may choose any $\nabla f(0)$ and $\nabla f(4)$ satisfying

$$-e^{-0} = -1 \leqslant \nabla f(0) \leqslant 0.25 \quad \text{and} \quad 0.25 \leqslant \nabla f(4) \leqslant 1 \ . \qquad \square$$

1.3 Positive definite matrices

Later on, when presenting necessary conditions for optimality in Section 1.5.2 and when discussing linear matrix inequalities (LMIs) in Chapter 6 we will encounter the notion of positive definite and positive semi-definite matrices.

Definition 1.12 (Positive definite matrix) *A matrix $A \in \mathbb{R}^{n \times n}$ is positive definite, denoted by $A > 0$, if $x^T Ax > 0$ for all $x \in \mathbb{R}^n$ with $x \neq 0$.*

Definition 1.13 (Positive semi-definite matrix) *A matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite, denoted by $A \geqslant 0$, if $x^T Ax \geqslant 0$ for all $x \in \mathbb{R}^n$.*

Likewise, we say that A is negative definite, denoted by $A < 0$, if $x^T Ax < 0$ for all $x \in \mathbb{R}^n$ with $x \neq 0$. If $x^T Ax < 0$ for some vectors $x \in \mathbb{R}^n$ and $y^T Ay > 0$ for some vectors $y \in \mathbb{R}^n$, then A is said to be indefinite.

One way to determine whether or not a symmetric matrix is positive definite, is via its principal minors. Let $A \in \mathbb{R}^{n\times n}$ be symmetric and consider a subset $\mathscr{I} \subseteq \{1,2,\dots,n\}$. The determinant of the submatrix $A_{\mathscr{I},\mathscr{I}}$ of A obtained by keeping only the rows and columns indexed by $\mathscr{I}$ is called a principal minor. If $\mathscr{I}$ is of the form $\{1,2,\dots,k\}$ with $k \leqslant n$ then the determinant of $A_{\mathscr{I},\mathscr{I}}$ is called a *leading* principal minor. Now A is positive definite if and only if all its leading principal minors of A are positive. For positive semi-definite matrices, *all* principal minors (not only the leading ones) have to be nonnegative.

For a symmetric matrix positive-definiteness can also be assessed via the eigenvalues of the matrix. If A is symmetric, then A is positive definite if and only if all its eigenvalues are positive (i.e., larger than 0). If A is symmetric, then A is positive semi-definite if and only if all its eigenvalues are nonnegative (i.e., larger than or equal to 0). If A is symmetric, then A is indefinite if and only if some of its eigenvalues are positive while some of its other eigenvalues are negative.

Furthermore, if $A, B > 0$ then $A^T > 0$ and $A + B > 0$. If $A > 0$ and if X is a non-singular matrix, then $XAX^T > 0$.

Example 1.14 The matrix $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ is positive definite. The matrix $B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ is positive semi-definite. The matrix $C = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ is indefinite. □

1.4 Optimization problems

Depending on the function f and on x, a distinction among optimization problems can be made:

- If x has real values:
  - If f is linear or affine[2], e.g., $f(x_1, x_2) = 3x_1 + 5x_2 + 6$, no (finite) minimum exists. Only if constraints are present, a minimum can be expected at a constraint. If the constraint functions are linear in the parameter vector x we deal with a linear programming problem. This will be discussed in Chapter 2.

    Linear programming problem:

    $$\begin{aligned} &\min_x c^T x \\ &\text{s.t. } Ax = b \\ &\qquad x \geqslant 0 \end{aligned} \qquad \text{or} \qquad \begin{aligned} &\min_x c^T x \\ &\text{s.t. } Ax \leqslant b \\ &\qquad x \geqslant 0 \end{aligned}$$

    where "s.t." is the abbreviation of "subject to".

[2]We say that a function f of $x \in \mathbb{R}^n$ is affine if it is of the form $f(x) = a_1x_1 + a_2x_2 + \cdots + a_nx_n + b = a^T x + b$ with $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. If $b = 0$ then we say that the function is linear. Note that some people (wrongly) also use the phrase "linear" even if $b \neq 0$. In fact, we will also adopt the latter convention in these lecture notes.

– If f is quadratic, e.g., $f(x_1,x_2) = 2x_1^2 + 3x_2^2 + 5x_1x_2 - 14x_1 - 17x_2 + 10$, and the constraint functions h and g (cf. the general expression (1.1)–(1.2) for a constrained optimization problem) are linear or affine, we deal with a quadratic programming problem. This will be discussed in Chapter 3.

Quadratic programming problem:

$$\begin{aligned} &\min_x \ \frac{1}{2}x^T Hx + c^T x \\ &\text{s.t. } Ax = b \\ &\qquad x \geqslant 0 \end{aligned} \qquad \text{or} \qquad \begin{aligned} &\min_x \ \frac{1}{2}x^T Hx + c^T x \\ &\text{s.t. } Ax \leqslant b \\ &\qquad x \geqslant 0 \end{aligned}$$

– If f is a nonlinear, non-convex function, then no general analytic solution is known. For this type of optimization problems several nonlinear optimization techniques are available, some of which will be discussed in Chapters 4 and 5. Some of these algorithms can use the analytic expressions for the gradient $\nabla f(x)$ and the Hessian $H(x)$ if they are available. There are two variants of this type of optimization problem: unconstrained and constrained:

Unconstrained nonlinear, non-convex optimization problem:

$$\min_x f(x)$$

where f is a nonlinear, non-convex function.

Constrained nonlinear, non-convex optimization problem:

$$\begin{aligned} &\min_x f(x) \\ &\text{s.t. } h(x) = 0 \\ &\qquad g(x) \leqslant 0 \end{aligned}$$

where in general f, h, g are nonlinear non-convex functions.

- If f is a convex function of the parameter vector x, robust convergent algorithms can be derived. Convex optimization algorithms are discussed in Chapter 6.

Convex optimization problem:

$$\begin{aligned}&\min_x f(x)\\&\text{s.t. } g(x) \leqslant 0\end{aligned}$$

where f, g are convex functions.

- If x has integer values, an integer optimization problem arises. Examples of integer parameters problems are routing/sequencing problems in batch processes, path-length problems, transportation and allocation problems, model order determination. If f is a nonlinear function of x, no general solution technique exists. Some techniques to solve the integer optimization problem are discussed in Chapter 11.

1.5 Conditions for optimality

In this section we treat the recognition of minima and maxima of the objective function for both unconstrained and constrained optimization problems. First, we will discuss necessary conditions for a minimum or a maximum. Next, we will also present necessary and sufficient conditions for the unconstrained and the convex case.

1.5.1 Necessary conditions for optimality

In this section we only give *necessary* conditions for a minimum or a maximum. We refer to the literature (e.g., [29, 37]) for *sufficient* conditions since they are rather complex, especially for constrained optimization problems.

Unconstrained optimization problem:

For an unconstrained optimization problem a *necessary* condition for a minimum or a maximum of the function f in x is that the gradient is zero in x:

Zero-gradient condition:

$$\nabla f(x) = 0$$

Equality constrained optimization problem:

For an equality constrained optimization problem *necessary* conditions for a minimum or maximum of the function f in x, satisfying $h(x) = 0$, are given by the Lagrange conditions:

Lagrange conditions:

There exists a vector λ, such that

$$\begin{aligned} \nabla f(x) + \nabla h(x)\,\lambda &= 0 \\ h(x) &= 0\ . \end{aligned}$$

Inequality/equality constrained optimization problem:

For an inequality/equality constrained optimization problem *necessary* conditions for a minimum or maximum of the function f in x, satisfying $h(x) = 0$ and $g(x) \leqslant 0$, are given by the Kuhn-Tucker conditions:

Kuhn-Tucker conditions:

There exist vectors λ and μ such that

$$\begin{aligned} \nabla f(x) + \nabla g(x)\,\mu + \nabla h(x)\,\lambda &= 0 \\ \mu^T g(x) &= 0 \\ \mu &\geqslant 0 \\ h(x) &= 0 \\ g(x) &\leqslant 0\ . \end{aligned}$$

1.5.2 Necessary and sufficient conditions for optimality

In this section we only give *necessary and sufficient* conditions for a minimum or a maximum of unconstrained and convex problems. We refer to the literature (e.g., [29, 37]) for conditions for constrained non-convex problem since they are rather complex.

Unconstrained optimization problem:

Recall that for an unconstrained optimization problem a necessary condition for a minimum or a maximum of the function f in x is that the gradient is zero in x: $\nabla f(x) = 0$. If in addition the Hessian is positive (semi)definite in x, then we have a local minimum.
If $\nabla f(x) = 0$ and $H(x)$ is negative (semi)definite, then x is a local maximum.
If $\nabla f(x) = 0$ and $H(x)$ is indefinite, then x is a so-called saddle point.

For a quadratic function the three cases above are represented in Figure 1.9. For all three cases the point under consideration is the origin, where the gradient ∇f equals 0, and the Hessian is respectively positive definite (Fig. 1.9(a)), negative definite (Fig. 1.9(b)), and indefinite (Fig. 1.9(c)).

Convex optimization problem:

For convex optimization problems the Kuhn-Tucker conditions given above give *necessary and sufficient* conditions for a minimum or maximum.

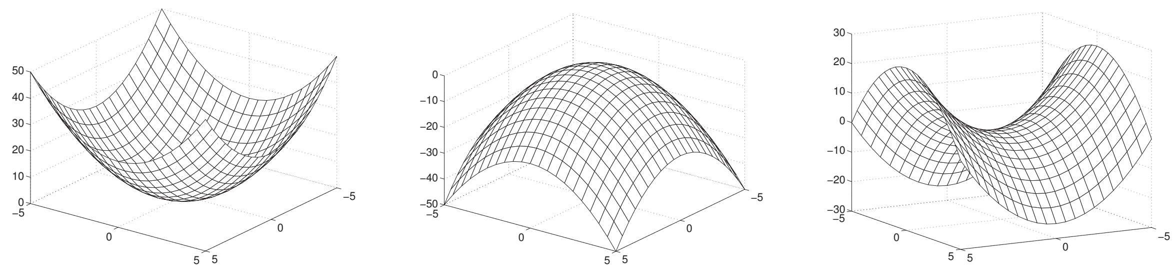


Figure 1.9: (a) Local minimum, (b) local maximum, and (c) saddle point.

1.6 Convergence and stopping criteria

Optimization techniques usually involve an iterative scheme, where in each iteration a new, and hopefully better, estimate of the parameter vector x is made. Some optimization techniques, like linear and quadratic programming, provide the optimal solution in a finite number of steps. When to stop the iteration can easily be determined by the optimality conditions of the previous section. Other techniques, like convex and nonlinear optimization can only find approximations of the optimum. An important property of these algorithms is whether they will converge to the optimal value x^* or not.

A measure for convergence is given by the following expression:

$$\beta = \lim_{k\to\infty} \frac{\| x_{k+1} - x^* \|_2}{\| x_k - x^* \|_2^p}$$

for some $p \in \mathbb{N}_0$, where x_k is the approximation obtained in the kth iteration step and $\|x\|_2 \overset{\text{def}}{=} \sqrt{x^T x}$ denotes the 2-norm of the vector x.

An algorithm is called linearly convergent if for $p = 1$ we find $0 < \beta < 1$. An algorithm is called super-linearly convergent if for $p = 1$ we find $\beta = 0$. An algorithm is called quadratically convergent if for $p = 2$ we find $0 < \beta < 1$.

If an algorithm converges, we still have to decide at what point the estimate is good enough. For convex optimization algorithms an upper bound for the measure $|\, f(x_k) - f(x^*) \,|$ can be derived. This implies that we can choose $\varepsilon_f > 0$, and stop the algorithm as soon as

$$|\, f(x_k) - f(x^*) \,| \leqslant \varepsilon_f \tag{1.3}$$

Moreover, for the ellipsoid method an upper bound for both $\| x_k - x^* \|_2$ and $|\, f(x_k) - f(x^*) \,|$ can be derived. So then we can choose $\varepsilon_x > 0$ or $\varepsilon_f > 0$, and stop the algorithm as soon as

$$\| x_k - x^* \|_2 \leqslant \varepsilon_x \qquad \text{or} \qquad |\, f(x_k) - f(x^*) \,| \leqslant \varepsilon_f \tag{1.4}$$

If constraints are present, then for the cutting-plane method and the ellipsoid method the criterion

$$g(x_k) \leqslant \varepsilon_g$$

with $\varepsilon_g > 0$ should also hold in addition to (1.3) or (1.4).
For unconstrained nonlinear optimization problems we can take the absolute value of the gradient as a measure, and stop the algorithm as soon as

$$\| \nabla f(x_k) \|_2 \leqslant \varepsilon_\nabla$$

For constrained nonlinear optimization problems we can take the absolute value of the Lagrange or Kuhn-Tucker formulas as a measure, and stop the algorithm as soon as

$$\begin{aligned} \| \nabla f(x_k) + \nabla g(x_k)\, \mu_k + \nabla h(x_k)\, \lambda_k \|_2 &\leqslant \varepsilon_{\mathrm{KT},1} \\ | \mu_k^T\, g(x_k) | &\leqslant \varepsilon_{\mathrm{KT},2} \\ \mu_k &\geqslant -\varepsilon_{\mathrm{KT},3} \\ \| h(x_k) \|_2 &\leqslant \varepsilon_{\mathrm{KT},4} \\ g(x_k) &\leqslant \varepsilon_{\mathrm{KT},5} \end{aligned}$$

where $\varepsilon_{\mathrm{KT},i} > 0$.

If no gradient or Jacobian is available, we have to use more heuristic stopping criteria, like:

$$\| x_{k+1} - x_k \|_2 \leqslant \varepsilon_x \qquad \text{or} \qquad | f(x_{k+1}) - f(x_k) | \leqslant \varepsilon_f$$

In addition, we usually also include an upper bound for the number of iteration steps in the stopping criterion.

Chapter 2

Linear Programming

Example 2.1 The brewery scheduling problem:
A small brewery can sell up to 100 boxes of beer per day, no matter if it is blond or dark beer. The brewing process can operate up to 14 hours a day. It takes 1 hour to produce 10 boxes of blond beer and it takes 2 hours to produce 10 boxes of dark beer. A box blond beer brings in 20 dollars, a box dark beer brings in 30 dollars. The objective function can be described as "profit", expressed in dollars, and must be maximized.

We can formulate the brewery scheduling problem as an optimization problem:

$$\max_x f(x) = \max_{x_1,x_2} \; 20x_1 + 30x_2$$

where x_1, x_2 denote the number of boxes of respectively blond and dark beer that are produced each day. The constraints are

$$\begin{array}{rcrcl} x_1 & + & x_2 & \leqslant & 100 \\ 0.1x_1 & + & 0.2x_2 & \leqslant & 14 \\ & & x_1, x_2 & \geqslant & 0 \end{array}$$

The first constraint shows that the production cannot exceed 100 boxes of beer per day. The second constraint reflects the fact that the brewer can only produce beer 14 hours a day. The last two non-negativity constraints are obvious because the number of boxes blond and dark beer cannot be smaller than zero.

Figure 2.1 gives a graphical representation of the problem. The four lines *AC*, *AB*, *BD* and *CD* correspond to the boundaries of the constraints: $x_1 = 0$, $x_2 = 0$, $x_1 + x_2 = 100$ and $0.1x_1 + 0.2x_2 = 14$. Only points (x_1, x_2) that belong to the inner area of the line segments *AC*, *CD*, *DB* and *BA* or that lie on the line segments themselves, are feasible. We have also plotted dotted lines for different levels of "profit", (a) for $f(x) = 1500$ dollars, (b) for $f(x) = 2400$ dollars and (c) for $f(x) = 3300$ dollars. In order to find the optimum we want to shift the dotted lines in a parallel way in the north-east direction, but such that there still is an intersection with the feasible set *ABCD*. This implies that the optimal solution is given by the point *D*, corresponding to

$$\begin{aligned} x_1^* &= 60 \text{ boxes of blond beer} \\ x_2^* &= 40 \text{ boxes of dark beer} \\ f(x^*) &= 20x_1^* + 30x_2^* = 2400 \text{ dollars} \;. \end{aligned}$$

□

Note that the optimal solution corresponds to a vertex of the feasible set *ABCD*. This can be generalized: the optimal solution of a linear programming solution, if it exists, can always be selected

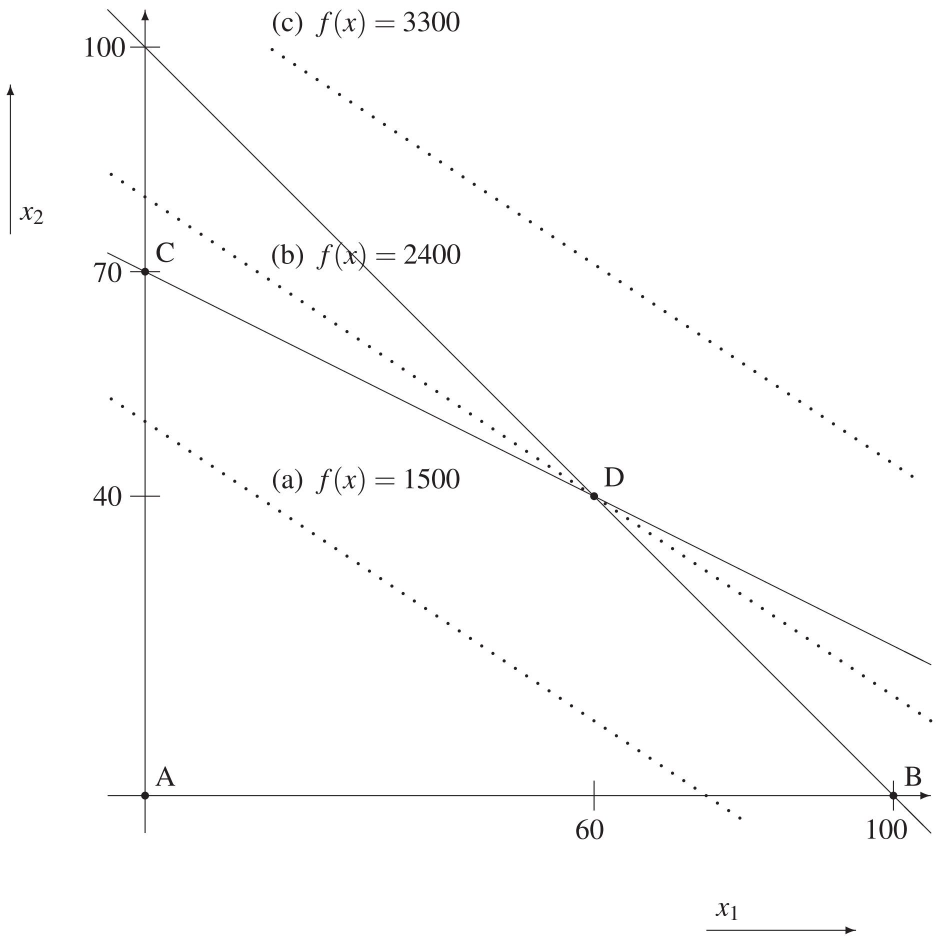


Figure 2.1: Brewery scheduling problem.

such that it coincides with a vertex of the feasible set. The graphical method of the above two-dimensional example could be extended to more dimensions. However, it then becomes difficult to make a nice graphical representation of the problem. Therefore, we will discuss in the next section an algebraic solution, which is based on matrix calculations and which can be implemented very easily on a computer.

2.1 The linear programming problem

Definition 2.2 (Linear Programming (LP) problem – Standard form)
The standard form of the linear programming problem is defined as follows: minimize the objective function

$$f(x_1, x_2, \ldots, x_n) = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

with respect to the linear constraints

$$\begin{array}{lclcl} a_{11}x_1 & +\cdots+ & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & +\cdots+ & a_{2n}x_n & = & b_2 \\ \vdots & & & & \vdots \\ a_{m1}x_1 & +\cdots+ & a_{mn}x_n & = & b_m \end{array}$$

and the non-negativity constraint

$$x_i \geqslant 0 \ \ \textit{for} \ \ i = 1, \ldots, n.$$

In matrix/vector notation we obtain: minimize the objective function

$$f(x) = c^T x$$

with respect to the linear constraint

$$Ax = b$$

and the non-negativity constraint

$$x \geqslant 0 \ .$$

Note that any linear optimization problem in which a linear (or affine) function is minimized or maximized subject to linear equality or inequality constraints can be recast as a linear programming problem in standard form using the following procedure:

- If we have an affine objective function of the form $c^T x + d$ with d a constant, then the position of the optimum will not change if we replace the objective function by $c^T x$.
  Furthermore, maximizing $c^T x$ is equivalent to minimizing $-c^T x$.
- An inequality constraint of the form $A_{i,\bullet} x \leqslant b_i$ (where $A_{i,\bullet}$ is the ith row of A) can be transformed into the constraints

  $$\begin{aligned} A_{i,\bullet} x + s_i &= b_i \\ s_i &\geqslant 0 \ . \end{aligned}$$

  The dummy variable s_i is called a slack variable.
- If there is no nonnegativity constraint for x_i, then we can introduce two dummy variables x_i^+ and x_i^- with $x_i^+, x_i^- \geqslant 0$ and eliminate x_i using the relation $x_i = x_i^+ - x_i^-$.

Example 2.3 We show how the brewery scheduling problem of Example 2.1 can be reformulated as a linear programming problem in standard form. We define the objective function as negative profit:

$$f(x) = -(20x_1 + 30x_2) \ .$$

The constraints

$$\begin{aligned} x_1 + \quad x_2 &\leqslant 100 \\ 0.1x_1 + 0.2x_2 &\leqslant 14 \\ x_1, x_2 &\geqslant 0 \end{aligned}$$

are equivalent to

$$\begin{aligned} x_1 + \quad x_2 + x_3 \quad &= 100 \\ 0.1x_1 + 0.2x_2 + \quad x_4 &= 14 \end{aligned}$$

provided that

$$x_1, x_2, x_3, x_4 \geqslant 0 \ .$$

Note that the slack variables x_3 and x_4 have been introduced to obtain equality constraints. If we define

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \quad c = \begin{bmatrix} -20 \\ -30 \\ 0 \\ 0 \end{bmatrix}, \quad b = \begin{bmatrix} 100 \\ 14 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0.1 & 0.2 & 0 & 1 \end{bmatrix} ,$$

we have formulated the brewery scheduling problem as a linear programming problem in standard form. □

2.2 The simplex method

In this section we give a rough outline of the *simplex method*, which can be used to solve linear programming problems.

Remark 2.4 The interior points methods that will be introduced in Chapter 6 can also be used to solve linear programming problems. Especially for large problems with more than 1000 variables or constraints interior points methods are in general more efficient than the simplex method. However, for small- or medium-scale problems we usually use the simplex method (or one of its variants). ◇

We assume without loss of generality that the $m \times n$ matrix A has full row rank m. Let B be a nonsingular $m \times m$ submatrix of A. The remaining part of A is denoted by N. Now the vectors x and c can be split in two parts x_B, x_N and c_B, c_N respectively, corresponding to the partitioning of A. The equality constraint can be reformulated as

$$Ax = Bx_B + Nx_N = b$$

and so we have

$$x_B = B^{-1}b - B^{-1}Nx_N$$

and

$$\begin{aligned} z \stackrel{\text{def}}{=} f(x) &= c^T x \\ &= c_B^T x_B + c_N^T x_N \\ &= c_B^T B^{-1} b + (c_N^T - c_B^T B^{-1} N) x_N \\ &= z_0 + p^T x_N \ . \end{aligned}$$

To each choice of partitioning of A in B and N there corresponds a "basic solution" x of the LP problem by setting $x_N = 0$, leading to the basic solution

$$x_B = B^{-1}b, \ \ x_N = 0$$

with the corresponding "basic cost"

$$z_0 = c_B^T B^{-1} b \ .$$

It can be shown that each vertex of the feasible set of the LP problem corresponds to a basic solution. In the simplex algorithm we will jump from feasible basic solution to feasible basic solution in a structured way until we finally obtain the optimal solution. Note that going from one basic solution to another corresponds to interchanging a column of B and a column of N.
Let us now briefly explain the basic procedure of (one possible variant of) the simplex method. For the sake of simplicity we assume that the initial basic solution is feasible[1]. The procedure for selecting the columns of B and N that should be interchanged is as follows:

- Selection of the column of N:
  We will select the column of N that yields the largest decrease in objective function. Recall that we have $z = z_0 + p^T x_N$. In the new basic solution one of the components of x_N will no longer be 0. In order to get the largest possible decrease in the objective function for a unit change in one of the components of x_N, we select the column i of N such that p_i has the largest negative value:

$$i = \arg\min\{\, p_i \,|\, p_i < 0 \,\} \ . \tag{2.1}$$

- Selection of the column of B:
  We will take the largest possible step that still yields a feasible solution. The main idea is that one of the components of the current basic solution x_B, in particular $(x_B)_j$ with j determined as indicated next, will become zero in the new basic solution, whereas one of the components of x_N, in particular $(x_N)_i$ with i determined as indicated above, will change from 0 to a positive number (indicated by ε below).
  Let $N_{\bullet,i}$ be the ith column of N where i is determined by (2.1) and define $y = B^{-1}N_{\bullet,i}$. So $By = N_{\bullet,i}$. Furthermore, since $Bx_B = b$, we have

$$B\big(x_B - \varepsilon y\big) + \varepsilon N_{\bullet,i} = Bx_B - \varepsilon N_{\bullet,i} + \varepsilon N_{\bullet,i} = b \tag{2.2}$$

  for any real number ε. In the new basic solution $(x_N)_i$ will be positive and one of the components of x_B will become 0. The new basic solution is given by $x_B - \varepsilon y$, i.e., we start from the current x_B and we take a step in the direction y. Now the step size (ε) should be determined. We will take the largest step that will still result in all components of the new basic solution to be nonnegative (recall that we have the constraint $x \geqslant 0$). Note that ε is positive. So if $y_j < 0$ (or $y_j = 0$) then $(x_B)_j$ will increase (or stay constant) and so there is no problem of $(x_B)_j$ becoming negative. If $y_j > 0$, then we have to select the largest ε such that $(x_B)_j - \varepsilon y_j \geqslant 0$ or $(x_B)_j \geqslant \varepsilon y_j$ or $\dfrac{(x_B)_j}{y_j} \geqslant \varepsilon$.
  This should hold for all j (with $y_j > 0$). Hence, we should select $\varepsilon = \min\left\{\dfrac{(x_B)_j}{y_j} | y_j > 0\right\}$, which yields

$$j = \arg\min\left\{\frac{(x_B)_j}{y_j} | y_j > 0\right\} \ . \tag{2.3}$$

[1] If necessary, a feasible basic solution can be determined by solving another LP problem for which a feasible solution is readily available. However, this is beyond the scope of these lecture notes.

If we now interchange column j of B and column i of N, we get again a feasible basic solution. In addition, the value of the objective function for the new basic solution will be lower than that for the previous basic solution. We continue until $p \geqslant 0$, since then no further decrease in the objective function can be obtained.

If we use other criteria to select the columns of i and j, we get another variant of the simplex method. For so-called anti-cycling rules that can be used to appropriately break ties in the choice of i and j in (2.1) and (2.3), for more complex rules for selecting the columns of B and N that have to be interchanged, and for a more elaborate treatment of the simplex method the interested reader is referred to [10, 29, 38, 45] and the references therein.

Note that the partitioning of A can only be made in a finite number of ways, and so only a finite number of basic solutions x_B exists and has to be evaluated. As a consequence, the simplex method always finds the optimal solution in a *finite number of steps*.

Example 2.5 Solution of the brewery scheduling problem using the simplex method:
We have

$$c = \begin{bmatrix} -20 \\ -30 \\ 0 \\ 0 \end{bmatrix}, \quad b = \begin{bmatrix} 100 \\ 14 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0.1 & 0.2 & 0 & 1 \end{bmatrix} .$$

Suppose our first choice of B and N is

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad N = \begin{bmatrix} 1 & 1 \\ 0.1 & 0.2 \end{bmatrix} .$$

Then we find x_B, x_N, c_B and c_N as

$$x_B = \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 100 \\ 14 \end{bmatrix}, \quad x_N = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad c_B = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad c_N = \begin{bmatrix} -20 \\ -30 \end{bmatrix} .$$

This corresponds to $x_1 = 0, x_2 = 0, x_3 = 100$ and $x_4 = 14$, which is a feasible basic solution, given by point A in Figure 2.1. The corresponding values of z_0 and p are:

$$z_0 = c_B^T B^{-1} b = 0, \quad p^T = c_N^T - c_B^T B^{-1} N = \begin{bmatrix} -20 & -30 \end{bmatrix} .$$

Since $p^T \ngeqslant 0$, the optimum is not found yet.

Since -30 is the largest negative component of p, we select the second column of N (so $i = 2$ in accordance to (2.1)). We have $y = B^{-1} N_{\bullet,2} = [1 \;\; 0.2]^T$. Since

$$\frac{(x_B)_1}{y_1} = \frac{100}{1} = 100 \quad \text{and} \quad \frac{(x_B)_2}{y_2} = \frac{14}{0.2} = 70 ,$$

we select the second column of B (so $j = 2$ in accordance to (2.3)). Now we interchange the second column of B and the second column of N, which leads to

$$B = \begin{bmatrix} 1 & 1 \\ 0 & 0.2 \end{bmatrix}, \quad N = \begin{bmatrix} 1 & 0 \\ 0.1 & 1 \end{bmatrix} .$$

Now we have

$$x_B = \begin{bmatrix} x_3 \\ x_2 \end{bmatrix} = \begin{bmatrix} 30 \\ 70 \end{bmatrix}, \quad x_N = \begin{bmatrix} x_1 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad c_B = \begin{bmatrix} 0 \\ -30 \end{bmatrix}, \quad c_N = \begin{bmatrix} -20 \\ 0 \end{bmatrix} .$$

This corresponds to $x_1 = 0$, $x_2 = 70$, $x_3 = 30$ and $x_4 = 0$, which is a feasible basic solution, given by point C in Figure 2.1. The corresponding values of z_0 and p are:

$$z_0 = c_B^T B^{-1} b = -2100, \quad p^T = c_N^T - c_B^T B^{-1} N = \begin{bmatrix} -5 & 150 \end{bmatrix} .$$

Since $p^T \ngeqslant 0$, the optimum is not found yet.

Equation (2.1) yields $i = 1$. Since $y = [0.5 \ \ 0.5]^T$, and

$$\frac{(x_B)_1}{y_1} = \frac{30}{0.5} = 60 \quad \text{and} \quad \frac{(x_B)_2}{y_2} = \frac{70}{0.5} = 140 \ ,$$

we have $j = 1$ by (2.3). So we interchange the first column of B and the first column of N. Now we have

$$B = \begin{bmatrix} 1 & 1 \\ 0.1 & 0.2 \end{bmatrix}, \quad N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} ,$$

and

$$x_B = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 60 \\ 40 \end{bmatrix}, \quad x_N = \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad c_B = \begin{bmatrix} -20 \\ -30 \end{bmatrix}, \quad c_N = \begin{bmatrix} 0 \\ 0 \end{bmatrix} .$$

This corresponds to $x_1 = 60$, $x_2 = 40$, $x_3 = 0$ and $x_4 = 0$, which is a feasible basic solution, given by point D in Figure 2.1. The corresponding values of z_0 and p are:

$$z_0 = c_B^T B^{-1} b = -2400, \quad p^T = c_N^T - c_B^T B^{-1} N = \begin{bmatrix} 10 & 100 \end{bmatrix} .$$

Since now we have $p^T \geqslant 0$, the optimum is found: the optimal solution is $(60, 40)$ and the corresponding profit is 2400. □

Chapter 3

Quadratic Programming

In this chapter we consider optimization problems with a quadratic objective function with linear constraints. Such problems are called quadratic programming (QP) problems. There are two types of QP problems:

Definition 3.1 (Quadratic programming problem – Type 1)
Minimize the objective function

$$f(x) = \frac{1}{2} x^T H x + c^T x$$

where H is a ***positive semi-definite***[1] *matrix, with respect to the linear inequality constraint*

$$Ax \leqslant b$$

and the non-negativity constraint

$$x \geqslant 0 \ .$$

Definition 3.2 (Quadratic programming problem – Type 2)
Minimize the objective function

$$f(x) = \frac{1}{2} x^T H x + c^T x$$

where H is a ***positive semi-definite*** *matrix, with respect to the linear equality constraint*

$$Ax = b$$

and the non-negativity constraint

$$x \geqslant 0 \ .$$

Remark 3.3 We may assume without loss of generality that the matrix H of the QP problem is symmetric. Indeed, if H would be a non-symmetric matrix, we can define

$$H_{\text{new}} = \frac{1}{2}(H + H^T) \ .$$

Substitution of H by this H_{new} does not change the values of the objective function. So we can use the symmetric matrix H_{new} in the objective function.
Note that the objective function f of the QPs defined above is convex since the matrix H is positive semi-definite. ◇

[1]Recall that for $H \in \mathbb{R}^{n \times n}$ is positive semi-definite if $x^T Hx \geqslant 0$ for all $x \in \mathbb{R}^n$.

In the remainder of this chapter we shall only consider problems of type 2. Note that this implies no loss of generality since a quadratic programming problem of type 1 can be converted into a quadratic programming problem of type 2 as follows. First we introduce a “slack” variable y such that

$$Ax + y = b \quad \text{where} \quad y \geqslant 0 \ .$$

If we define

$$\bar{H} = \begin{bmatrix} H & 0 \\ 0 & 0 \end{bmatrix} \ , \ \bar{c} = \begin{bmatrix} c \\ 0 \end{bmatrix} \ , \ \bar{A} = \begin{bmatrix} A & I \end{bmatrix} \ , \ \bar{x} = \begin{bmatrix} x \\ y \end{bmatrix} \ ,$$

we obtain a QP problem of type 2:

$$\begin{aligned} &\min_{\bar{x}} \ \frac{1}{2}\bar{x}^T\bar{H}\bar{x} + \bar{c}^T\bar{x} \\ &\text{s.t.} \ \bar{A}\bar{x} = b \\ &\qquad \bar{x} \geqslant 0 \ . \end{aligned}$$

The unconstrained quadratic programming problem

If the constraints are omitted and only the quadratic objective function is considered, an analytic solution exists: An extremum of the objective function

$$f(x) = \frac{1}{2}x^T H x + c^T x$$

is found when the gradient ∇f is equal to zero

$$\nabla f(x) = Hx + c = 0$$

If H is non-singular, the extremum is reached for

$$x = -H^{-1}c$$

Because H is now a positive definite[2] matrix, this extremum will be a minimum (cf. Section 1.5.2).

3.1 Quadratic programming algorithm

Consider the QP problem of type 2 as defined in Definition 3.2. For this problem the necessary conditions for an optimum — i.e., the Kuhn-Tucker conditions — are (cf. p. 15):

$$Ax = b \tag{3.1}$$
$$Hx + A^T\lambda - \mu = -c \tag{3.2}$$
$$x, \mu \geqslant 0 \tag{3.3}$$
$$x^T\mu = 0 \ . \tag{3.4}$$

Note that the QP problems defined in Definitions 3.1 and 3.2 are convex since we only consider positive semi-definite matrices H. This implies that for these QP problems the Kuhn-Tucker conditions (3.1)–(3.4) are necessary and sufficient conditions for a minimum (cf. Section 1.5.2).

[2]In general, we consider positive semi-definite symmetric matrices H in this chapter. Such matrices have nonnegative eigenvalues. However, in the case considered here H is non-singular, which implies that its eigenvalues are all different from 0. Hence, H cannot be positive *semi*-definite, but it has to be a positive definite matrix.

Let us compare (3.1)–(3.4) with an LP problem. We recognize equality constraints (3.1)–(3.2) and a non-negativity constraint (3.3), two ingredients of an LP problem. However, we miss an objective function, and we have an additional nonlinear equality constraint (3.4) (Note that the multiplication $x^T\,\mu$ is nonlinear in the new parameter vector $[x^T\;\mu^T\;\lambda^T]^T$.) We can obtain a minimization problem by introducing two slack variables u_1 and u_2 and a linear objective function $\sum_i (u_1)_i + \sum_j (u_2)_j$. This leads to the following problem (where we assume for the sake of simplicity that $b \geqslant 0$ and $c \leqslant 0$ (See also Remark 3.4)):

$$\min_{x,\mu,\lambda,u_1,u_2} \sum_i (u_1)_i + \sum_j (u_2)_j \tag{3.5}$$

$$\text{s.t.} \qquad Ax + u_1 = b \tag{3.6}$$

$$Hx + A^T\lambda - \mu + u_2 = -c \tag{3.7}$$

$$x\,,\,\mu\,,\,u_1\,,\,u_2 \geqslant 0 \tag{3.8}$$

$$x^T\,\mu = 0 \tag{3.9}$$

Note that we can easily obtain a feasible solution for this problem by selecting $u_1 = b$, $u_2 = -c$, $x = 0$, $\lambda = 0$ and $\mu = 0$. If we construct

$$A_0 = \begin{bmatrix} A & 0 & 0 & I & 0 \\ H & A^T & -I & 0 & I \end{bmatrix}, \quad b_0 = \begin{bmatrix} b \\ -c \end{bmatrix}, \quad c_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \quad x_0 = \begin{bmatrix} x \\ \lambda \\ \mu \\ u_1 \\ u_2 \end{bmatrix}, \tag{3.10}$$

where the 0 and 1 entries in c_0 represent column vectors with all entries equal to 0 or 1 that have the same size as x, λ, μ, u_1 and u_2 respectively, the problem can be written as:

$$\min c_0^T x_0 \quad \text{subject to} \quad A_0 x_0 = b_0 \text{ and } x_0 \geqslant 0$$

with the additional nonlinear constraint $x^T\,\mu = 0$.

It may be clear from the Kuhn-Tucker conditions in (3.1)–(3.4) that an optimum is only reached when u_1 and u_2 in (3.6)–(3.8) become zero. This can be achieved by using a simplex algorithm which is modified in the sense that there is an extra condition on finding feasible basic solutions (see Chapter 2 for the simplex algorithm), namely $x^T\,\mu = 0$. Note from (3.3) and (3.4) that

$$x^T\,\mu = x_1\,\mu_1 + x_2\,\mu_2 + \ldots + x_n\,\mu_n = 0 \text{ where } x_i,\,\mu_i \geqslant 0 \text{ for } i = 1,\ldots,n$$

means that the nonlinear constraint (3.4) can be rewritten as:

$$x_i\,\mu_i = 0 \text{ for } i = 1,\ldots,n$$

and thus either $x_i = 0$ or $\mu_i = 0$. The extra feasibility condition on a basic solution now becomes $x_i\,\mu_i = 0$ for $i = 1,\ldots,n$. Now we can compute the optimal solution of the QP problem using the modified simplex algorithm. Note that this algorithm will also always find the optimal solution in a *finite number of steps*.

Remark 3.4 In the above derivations we assumed for the sake of simplicity that $b \geqslant 0$ and $c \leqslant 0$. If this is not the case, we have to replace u_1 in (3.6) by $D(b)\,u_1$ and u_2 in (3.7) by $D(-c)\,u_2$ where $D(b)$ is a diagonal matrix with $\big(D(b)\big)_{ii} = 1$ if $b_i \geqslant 0$ and $\big(D(b)\big)_{ii} = -1$ if $b_i < 0$. ◇

We will illustrate the modified simplex algorithm on the basis of an example.

Example 3.5 Consider the following QP problem of type 2:

$$\begin{aligned} &\min_x \frac{1}{2} x^T H x + c^T x \\ &\text{s.t. } Ax = b \\ &\qquad x \geqslant 0 \end{aligned}$$

where

$$H = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ 0 & 1 & 4 & 2 \\ 0 & 0 & 2 & 4 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ -3 \\ -2 \\ 0 \end{bmatrix}, \quad A = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 0 & 2 & 1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 4 \\ 4 \end{bmatrix}.$$

Before we apply the modified simplex algorithm, we check if a simple solution can be found. If we minimize the unconstrained function

$$\min_x \frac{1}{2} x^T H x + c^T x$$

we obtain $x = \begin{bmatrix} 7 & 14 & -4 & 2 \end{bmatrix}^T$. This solution is not feasible, because neither the equality constraint $Ax = b$, nor the non-negativity constraint $x \geqslant 0$ are satisfied.

We construct the matrix A_0, the vectors b_0, c_0 and the parameter vector x_0 according to equation (3.10).

Recall that selecting

$$x = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \lambda = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \mu = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad u_1 = b = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, \quad u_2 = -c = \begin{bmatrix} 0 \\ 3 \\ 2 \\ 0 \end{bmatrix}$$

yields a feasible initial solution. This solution corresponds to selecting the six rightmost columns of A_0 as the columns of B. However, the optimum is not found yet, because $u_1, u_2 > 0$.

After a finite number of iterations the optimum is found by selecting columns 1, 2, 5, 6, 9 and 10 of the matrix A_0 for the matrix B. We find the optimal solution

$$x = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix}, \quad \lambda = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \mu = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \end{bmatrix}, \quad u_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad u_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

and the Kuhn-Tucker conditions (3.1)–(3.4) are satisfied for these values. □

Algorithms that use a modified version of the simplex method are Wolfe's algorithm [46] and the pivoting algorithm of Lemke [26].

3.2 System identification example

In this section we consider a simple system identification example. The purpose of this example is to show that QP problems also arise in system identification problems. More details are given in the course *Filtering and Identification* (SC4040).

Example 3.6 Consider a discrete-time ARX (Auto Regressive eXogenous input) model with input signal u, output signal y and noise signal e:

$$y(n+1)+ay(n)=bu(n)+e(n) \quad .$$

Suppose that we have the measured input and output signals

$$\begin{bmatrix} y(1) \\ y(2) \\ y(3) \\ y(4) \\ y(5) \end{bmatrix} = \begin{bmatrix} 0 \\ 0.74 \\ -0.80 \\ 1.97 \\ -1.09 \end{bmatrix} \qquad \begin{bmatrix} u(1) \\ u(2) \\ u(3) \\ u(4) \end{bmatrix} = \begin{bmatrix} 0.33 \\ -0.12 \\ 0.80 \\ -0.07 \end{bmatrix}$$

The following equality will hold:

$$\begin{bmatrix} y(2) \\ y(3) \\ y(4) \\ y(5) \end{bmatrix} + \begin{bmatrix} y(1) & -u(1) \\ y(2) & -u(2) \\ y(3) & -u(3) \\ y(4) & -u(4) \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} e(1) \\ e(2) \\ e(3) \\ e(4) \end{bmatrix}$$

Of course, the signal e is unknown. Therefore, we are unable to retrieve the true parameters a and b and we can only find estimates $\hat{a}$ and $\hat{b}$:

$$\begin{bmatrix} y(2) \\ y(3) \\ y(4) \\ y(5) \end{bmatrix} - \begin{bmatrix} -y(1) & u(1) \\ -y(2) & u(2) \\ -y(3) & u(3) \\ -y(4) & u(4) \end{bmatrix} \begin{bmatrix} \hat{a} \\ \hat{b} \end{bmatrix} = \begin{bmatrix} \varepsilon(1) \\ \varepsilon(2) \\ \varepsilon(3) \\ \varepsilon(4) \end{bmatrix}$$

where the signal ε describes or approximates the noise signal e. In matrix notation this becomes

$$Y-\Phi x=E$$

where $x=\begin{bmatrix} \hat{a} \\ \hat{b} \end{bmatrix}$ is the parameter vector. If we assume e to be zero-mean white noise, the best estimates $\hat{a}$ and $\hat{b}$ for the parameters a and b can be obtained by minimizing the energy of the signal ε, and we obtain the following problem:

$$\begin{aligned} \min_x \sum_{n=1}^{4} \varepsilon^2(n) &= \min_x E^T E = \min_x (Y-\Phi x)^T (Y-\Phi x) \\ &= \min_x \frac{1}{2} x^T H x + c^T x + d \end{aligned}$$

for $H = 2\Phi^T\Phi$, $c = -2\Phi^T Y$, and $d = Y^T Y$. We can drop the constant term d, since it does not influence the position of the optimum. As a consequence, we obtain the following minimization problem:

$$\begin{aligned}\min_x f(x) &= \min_x \frac{1}{2} x^T H x + c^T x \\ &= \min_x x^T \begin{bmatrix} 5.068 & 0.867 \\ 0.867 & 0.768 \end{bmatrix} x + \begin{bmatrix} -8.630 & -3.985 \end{bmatrix} x \ .\end{aligned}$$

Of course, we like the model to be stable[3] and therefore we add the additional constraint $-0.99 \leqslant \hat{a} \leqslant 0.99$. This can be formulated as a linear inequality constraint:

$$Ax \leqslant b$$

with

$$A = \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix} \text{ and } b = \begin{bmatrix} 0.99 \\ 0.99 \end{bmatrix} \ .$$

We now have a quadratic programming problem of type 1 and we obtain the optimum $f(x^*) = -6.213$ for $x^* = \begin{bmatrix} 0.505 & 2.024 \end{bmatrix}^T$. The corresponding energy of the signal ε is given by

$$\sum_{n=1}^{4} \varepsilon^2(n) = f(x^*) + d = f(x^*) + Y^T Y = 0.0438 \ .$$

Note that the problem considered in this example can also be solved using a constrained linear least squares method [25]. $\square$

[3]The condition for stability is $-1 < \hat{a} < 1$, but in order to guarantee a certain margin of stability we impose a slightly stronger condition.

Chapter 4

Nonlinear Optimization without Constraints

In this chapter we consider unconstrained nonlinear optimization problems of the form

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

where f is a nonlinear non-convex function. We will present three classes of methods to solve such unconstrained nonlinear optimization problems:

1. **Newton and Quasi-Newton methods**

   These methods are based on a quadratic approximation of the objective function. In addition to the evaluation of the objective function, the Newton method and the Levenberg-Marquardt method require the evaluation of the Hessian and the gradient of the objective function. In quasi-Newton methods the Hessian is not computed explicitly but approximated; quasi-Newton methods only require the evaluation of the objective function and the gradient.

2. **Methods with direction determination and line search**

   Each iteration step of these methods consists of a direction determination followed by a line search, i.e., an optimization along this direction. The main idea is to choose a search direction d_k at an initial point x_k, and then to minimize the objective function $f(x)$ over the line

   $$x = x_k + d_k s \quad , \quad s \in \mathbb{R} \ .$$

   We obtain a one-dimensional minimization problem

   $$\min_s f(x_k + d_k s).$$

   The starting point for the next iteration is then given by

   $$x_{k+1} = x_k + d_k s_k^*$$

   where

   $$s_k^* = \arg\min_s f(x_k + d_k s) \ .$$

3. **Nelder-Mead Method**

   This method is based on the geometric properties of a simplex. The method is simple and only requires evaluations of the objective function without any knowledge of the gradient.

4.1 Newton and quasi-Newton methods

4.1.1 Newton's algorithm

Consider the second-order Taylor series expansion of the objective function $f(x)$ around an estimate x_0:

$$f(x) = f(x_0) + \nabla^T f(x_0)\,(x-x_0) + \frac{1}{2}(x-x_0)^T H(x_0)\,(x-x_0) + \mathscr{O}(\|x-x_0\|_2^3)$$

where $\nabla f(x_0)$ is the gradient of $f(x)$ in x_0 and $H(x_0)$ is the Hessian of $f(x)$ in x_0. The remaining error $\mathscr{O}(\|x-x_0\|_2^3)$ contains third and higher order terms of the difference $\|x-x_0\|_2$. The function $\tilde{f}$ defined by

$$\tilde{f}(x) = f(x_0) + \nabla^T f(x_0)(x-x_0) + \frac{1}{2}\,(x-x_0)^T H(x_0)(x-x_0)$$

is almost equal to the objective function f in the immediate neighborhood of x_0 (i.e., for $\|x-x_0\|_2$ small enough). Since minimizing $\tilde{f}$ is an unconstrained quadratic problem (cf. Chapter 3), a very good estimate for the optimum of the original problem is given by:

$$(x-x_0)_{\text{opt}} = -H^{-1}(x_0)\,\nabla f(x_0)$$

or

$$x_{\text{opt}} = x_0 - H^{-1}(x_0)\,\nabla f(x_0) \quad .$$

To improve the result we can repeat this procedure and so we obtain Newton's algorithm, which is characterized by the following iteration step:

$$x_{k+1} = x_k - H^{-1}(x_k)\,\nabla f(x_k) \quad .$$

The estimate x_{k+1} will be good if the error $\mathscr{O}(\|\,x_{k+1}-x_k\|_2^3)$ is not too big. The closer we are to the optimum x^*, the smaller the difference $\|x_{k+1}-x_k\|_2$. This means that, if the initial guess x_0 is close enough to x^*, the approximation will become better each step, and the algorithm converges.

4.1.2 Levenberg-Marquardt algorithm

The Newton algorithm does not always perform well, e.g., the inversion of the Hessian matrix may become numerically ill-conditioned if the Hessian matrix is almost singular. The Levenberg-Marquardt algorithm provides a solution for this problem by using

$$\hat{H}_k = \hat{H}_k(x_k) = \lambda I + H(x_k)$$

instead of $H(x_k)$. So we obtain

$$x_{k+1} = x_k - \hat{H}_k^{-1}\,\nabla f(x_k) = x_k - \Big(\lambda I + H(x_k)\Big)^{-1}\,\nabla f(x_k) \quad .$$

If $\|H(x_k)\| \gg \lambda$ the method will show the nice features of the Newton algorithm. However, if the norm of the Hessian is small compared to λ, the algorithm will take a step in the direction of the negative gradient (i.e., the steepest descent direction) and determine the new point as follows:

$$x_{k+1} \approx x_k - \lambda^{-1}\,\nabla f(x_k) \quad .$$

So by selecting an appropriate value for λ we can avoid numerical problems related to the inversion of the Hessian.

4.1.3 Quasi-Newton algorithms

Another problem of the Newton algorithm is that the computation of the Hessian matrix $H(x_k)$ is time-consuming. Quasi-Newton algorithms provide solutions for this problem by using an approximation of the Hessian. The two most widely used methods are:

1. The Broyden-Fletcher-Goldfarb-Shanno method

In this method we use an approximation of the Hessian based on the Broyden-Fletcher-Goldfarb-Shanno formula:

$$\hat{H}_k = \hat{H}_{k-1} + \frac{q_k q_k^T}{q_k^T s_k} - \frac{\hat{H}_{k-1} s_k s_k^T \hat{H}_{k-1}^T}{s_k^T \hat{H}_{k-1} s_k}$$

where

$$\begin{aligned} s_k &= x_k - x_{k-1} \\ q_k &= \nabla f(x_k) - \nabla f(x_{k-1}) \end{aligned}$$

and $\hat{H}_0 = H(x_0)$. So the iteration step becomes

$$x_{k+1} = x_k - \hat{H}_k^{-1} \nabla f(x_k) \quad .$$

2. The Davidon-Fletcher-Powell method

The Broyden-Fletcher-Goldfarb-Shanno formula gives a nice approximation of the Hessian. The main problem is that there is still an inversion involved. This can be avoided by using the Davidon-Fletcher-Powell formula, which gives an estimate of the inverse of the Hessian:

$$\hat{D}_k = \hat{D}_{k-1} + \frac{s_k s_k^T}{q_k^T s_k} - \frac{\hat{D}_{k-1} q_k q_k^T \hat{D}_{k-1}^T}{q_k^T \hat{D}_{k-1} q_k}$$

where

$$\begin{aligned} s_k &= x_k - x_{k-1} \\ q_k &= \nabla f(x_k) - \nabla f(x_{k-1}) \quad . \end{aligned}$$

and $\hat{D}_0 = H^{-1}(x_0)$. We obtain the following iteration step:

$$x_{k+1} = x_k - \hat{D}_k \nabla f(x_k) \quad .$$

4.1.4 Nonlinear least squares problems

In this section we consider an important class of optimization problems, namely the nonlinear least squares problem, which appear in many engineering problems where we want to minimize, e.g., the difference between two signals, or the energy of a signal. In general, a nonlinear least squares problem is defined as

$$\min_x f(x) = \min_x \|e(x)\|_2^2 = \min_x \sum_{i=1}^N e_i^2(x)$$

where the "error vector" $e(x)$ is given by

$$e(x) = \begin{bmatrix} e_1(x) & e_2(x) & \dots & e_N(x) \end{bmatrix}^T \quad .$$

Let the Jacobian of $e(x)$ be defined as

$$\nabla e(x) = \begin{bmatrix} \dfrac{\partial e_1}{\partial x_1} & \dfrac{\partial e_2}{\partial x_1} & \cdots & \dfrac{\partial e_N}{\partial x_1} \\ \dfrac{\partial e_1}{\partial x_2} & \dfrac{\partial e_2}{\partial x_2} & \cdots & \dfrac{\partial e_N}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial e_1}{\partial x_n} & \dfrac{\partial e_2}{\partial x_n} & \cdots & \dfrac{\partial e_N}{\partial x_n} \end{bmatrix} .$$

Since $f(x) = \|e(x)\|_2^2 = e^T(x)\,e(x)$, the gradient of f is given by

$$\nabla f(x) = 2\,\nabla e(x)\,e(x)$$

and the Hessian is given by

$$H(x) = 2\,\nabla e(x)\,\nabla^T e(x) + \sum_{i=1}^{N} 2\,\nabla^2 e_i(x) e_i(x)$$

where $\nabla^2 e_i$ is the Hessian of e_i.
When the second term is small (which will be the case if we are close to the optimum since then $e(x) \approx 0$), the Hessian can be approximated by

$$\bar{H}(x) = 2\,\nabla e(x)\,\nabla^T e(x) \ . \tag{4.1}$$

If we use this approximation in the Newton algorithm, we obtain the *Gauss-Newton algorithm*, which uses the following iteration formula:

$$x_{k+1} = x_k - \left(\nabla e(x_k)\,\nabla^T e(x_k)\right)^{-1} \nabla e(x_k)\,e(x_k) \ .$$

If we use approximation (4.1) in the Levenberg-Marquardt algorithm, the iteration formula becomes

$$x_{k+1} = x_k - \left(\frac{\lambda}{2} I + \nabla e(x_k)\,\nabla^T e(x_k)\right)^{-1} \nabla e(x_k)\,e(x_k) \ .$$

4.2 Methods with direction determination and line search

In this section we consider iterative optimization algorithms where in each iteration step we first select a search direction and then we perform a line search, i.e., an optimization along this search direction. We will first discuss the line search step.

4.2.1 Line search

Suppose that we have an initial point x_k and that we have already chosen a search direction d_k (see Section 4.2.2 for the determination of the search direction). Now we discuss how the following *line search* problem can be solved:

$$\min_s f(x_k + d_k s) = \min_s \bar{f}_k(s) \ .$$

So we want to determine the minimal function value for points lying on the line with direction d_k that passes through x_k. Therefore, we call this procedure “line search”; the resulting minimum is called “line minimum”. Note that the n-dimensional minimization of $f(x)$, $x \in \mathbb{R}^n$ is replaced by a one-dimensional minimization of $\bar{f}_k(s)$, $s \in \mathbb{R}$.

There exist several techniques to perform the line search, like fixed or variable step methods, parabolic or cubic interpolation methods, the golden section method or the Fibonacci method. These six methods will be discussed next.

Fixed-step method:

In the fixed-step line search procedure we take steps of length Δs, so $s = l\,\Delta s$, $l \in \mathbb{N}_0$. Assuming that d_k gives a descent direction, we increase l until

$$\bar{f}_k\,(l^*\,\Delta s) < \bar{f}_k\,((l^*+1)\,\Delta s)$$

This means that we have found a line minimum for

$$x_{k+1} = x_k + d_k\,l^*\,\Delta s$$

Variable step method:

The line search using fixed step-size Δs may need many steps to find the minimum, especially when the step size is chosen small and the optimum is far from the starting point x_k. In the variable step method we may vary the step size Δs_l for each step $l \in \mathbb{N}_0$ in order to speed up the search for the line minimum. If we find the line minimum in the l^*th step, the new parameter vector becomes

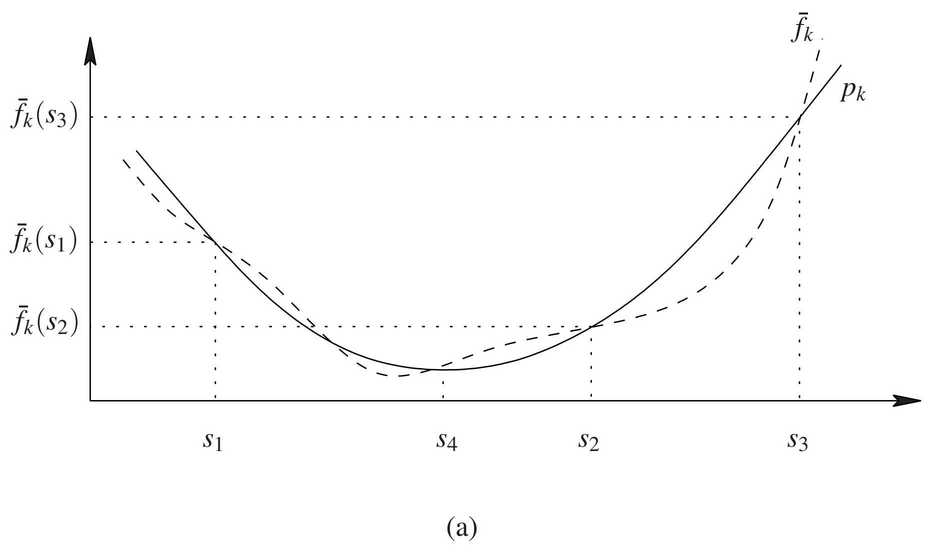
$$x_{k+1} = x_k + d_k \sum_{l=1}^{l^*} \Delta s_l$$

Parabolic interpolation:

Suppose we have three values s_1, s_2 and s_3 with the corresponding values of the objective function $\bar{f}_k(s_1)$, $\bar{f}_k(s_2)$ and $\bar{f}_k(s_3)$. A parabola p_k can be constructed through the points $(s_1, \bar{f}_k(s_1))$, $(s_2, \bar{f}_k(s_2))$ and $(s_3, \bar{f}_k(s_3))$, and the minimum s_4 of this parabola can be computed (see Figure 4.1 (a)) . The same procedure can be done using two function values $\bar{f}_k(s_1)$ and $\bar{f}_k(s_2)$ and the value of the derivative $\frac{d\bar{f}_k}{ds}(s_1)$ (see Figure 4.1 (b)). Now we can replace the point $(s_1, \bar{f}_k(s_1))$ by $(s_4, \bar{f}_k(s_4))$ and repeat the construction of the parabola. We continue until the line minimum is found or approximated accurately enough.

Cubic interpolation:

Using four function values $\bar{f}_k(s_1)$, $\bar{f}_k(s_2)$, $\bar{f}_k(s_3)$, and $\bar{f}_k(s_4)$, or using two function values $\bar{f}_k(s_1)$, $\bar{f}_k(s_2)$ and two derivatives $\frac{d\bar{f}_k}{ds}(s_1)$, $\frac{d\bar{f}_k}{ds}(s_2)$, a cubic function can be constructed through these points and the minimum s_5 of this cubic function can be computed. We repeat this construction of a cubic function until the line minimum is found.



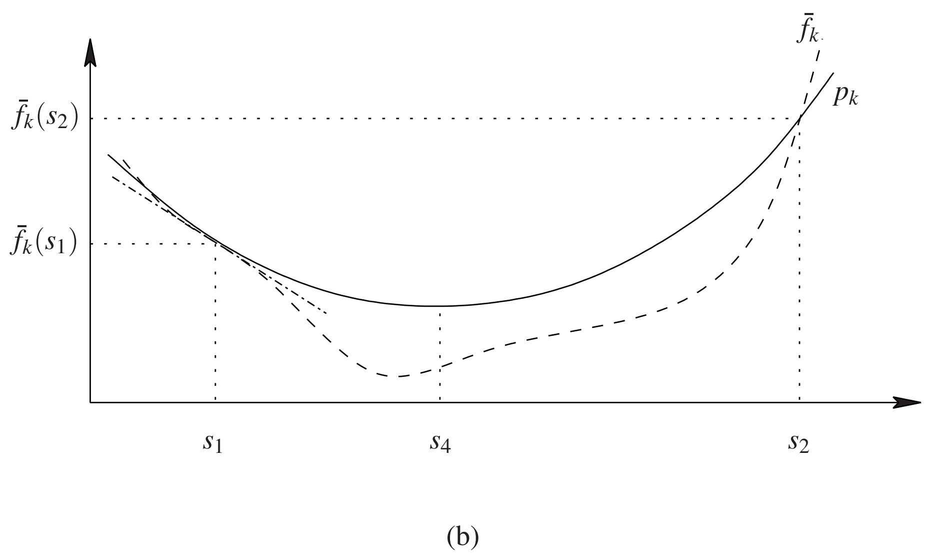


Figure 4.1: Parabolic interpolation of the function $\bar{f}_k$ (dashed curve): (a) using three function values, (b) using two function values and a derivative.

Golden section method:

Suppose we have two values $a_l = s_1$ and $d_l = s_2$ where $a_l < d_l$ with the corresponding values of the objective function $\bar{f}_k(a_l)$ and $\bar{f}_k(d_l)$. We assume that the function $\bar{f}_k$ is strictly unimodal on $[a_l, d_l]$. So there is exactly one local minimum s^* in the interval $[a_l, d_l]$ and the function $\bar{f}_k$ is (strictly) decreasing on $[a_l, s^*]$ and (strictly) increasing on $[s^*, d_l]$. We can construct three sub-intervals of $[a_l, d_l]$ by choosing

$$b_l = \lambda\, a_l + (1-\lambda)\, d_l$$

$$c_l = (1-\lambda)\, a_l + \lambda\, d_l$$

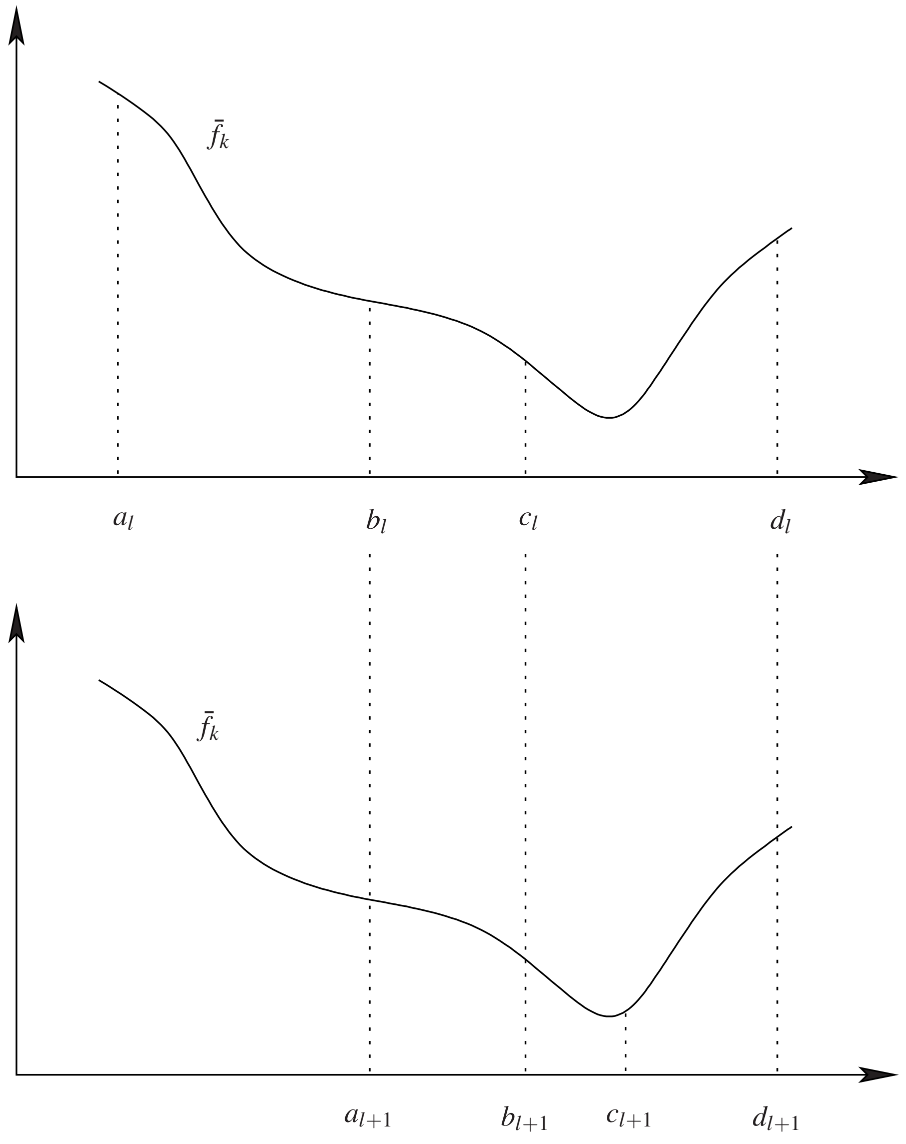


Figure 4.2: Golden section method.

for some λ satisfying $0.5 < \lambda < 1$. Note that $a_l < b_l < c_l < d_l$.
Now consider two cases[1]:

- If $\bar{f}_k(b_l) > \bar{f}_k(c_l)$, we know that the minimum must be in the interval $[b_l, d_l]$. We can define $a_{l+1} = b_l$, $d_{l+1} = d_l$ and compute $b_{l+1} = \lambda\, a_{l+1} + (1-\lambda)\, d_{l+1}$ and $c_{l+1} = (1-\lambda)\, a_{l+1} + \lambda\, d_{l+1}$.
- If $\bar{f}_k(b_l) < \bar{f}_k(c_l)$, we know that the minimum must be in the interval $[a_l, c_l]$. We can define $a_{l+1} = a_l$, $d_{l+1} = c_l$ and compute $b_{l+1} = \lambda\, a_{l+1} + (1-\lambda)\, d_{l+1}$ and $c_{l+1} = (1-\lambda)\, a_{l+1} + \lambda\, d_{l+1}$.

Now we choose $\lambda = \frac{1}{2}(\sqrt{5}-1) \approx 0.6180$, which is called the golden section constant because it preserves certain ratios. This ratio property can be exploited by observing that for $\bar{f}_k(b_l) > \bar{f}_k(c_l)$ we

[1]Since the function f is assumed to be strictly unimodal, the case $\bar{f}_k(b_l) = \bar{f}_k(c_l)$ implies that the minimum is in the interval $[b_l, c_l]$. However, for the sake of simplicity we include this case in the case $\bar{f}_k(b_l) > \bar{f}_k(c_l)$.

find $b_{l+1} = c_l$, and for $\bar{f}_k(b_l) < \bar{f}_k(c_l)$ we find $c_{l+1} = b_l$. This means that only one new function value has to be calculated in each step (see Figure 4.2). As a consequence, the computation time is reduced.

Fibonacci method:

The Fibonacci method is a modification of the golden section method in which the constant factor λ is changed for each iteration as explained next.

The sequence $\{\mu_k\}_{k=0}^{\infty} = 0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\dots$ is called the Fibonacci sequence. This sequence is generated by the recursion equation $\mu_k = \mu_{k-1} + \mu_{k-2}$ with the initial conditions $\mu_0 = 0$ and $\mu_1 = 1$.

Assume we have generated the numbers of Fibonacci sequence until $k = n$ (see below for how to determine n). If for computing the new points b_{l+1} and c_{l+1} we select

$$\lambda_{l+1} = \frac{\mu_{n-1-l}}{\mu_{n-l}} \quad ,$$

then it can be shown that we can reuse points from one iteration to the next. So in the Fibonacci method also only one new function value has to be calculated in each step. Furthermore, it can be shown that among the function comparison methods the Fibonacci methods gives the optimal interval reduction for a given number of function evaluations [37].

When applying the Fibonacci method, the $(l+1)$st subinterval $[a_{l+1}, d_{l+1}]$ is obtained by reducing the length of the lth subinterval by a factor λ_{l+1}. Note that the last step will therefore be for $\lambda_{n-2} = \dfrac{\mu_2}{\mu_3} = \dfrac{1}{2}$, i.e., in the last step the interval is divided in two, and the algorithm terminates (as $b_{n-2} = c_{n-2}$). Hence, if we start with an initial interval $[a_0, b_0]$, then the length of the final interval will be

$$\lambda_1\, \lambda_2 \dots \lambda_{n-2}(b_0 - a_0) = \frac{\mu_{n-1}}{\mu_n}\,\frac{\mu_{n-2}}{\mu_{n-1}} \dots \frac{\mu_2}{\mu_3}(b_0 - a_0) = \frac{\mu_2}{\mu_n}(b_0 - a_0) = \frac{1}{\mu_n}(b_0 - a_0)$$

So if we want a final interval length that is smaller than a given tolerance $\varepsilon > 0$, then we have to select n such that $\frac{1}{\mu_n}(b_0 - a_0) \leqslant \varepsilon$.

4.2.2 Determination of the search direction

For the determination of the search direction d_k we consider two different methods:

- Perpendicular search methods
- Gradient methods and conjugate-gradient methods

The perpendicular search methods are used if the gradient $\nabla f(x)$ of the objective function is not available or if we do not want to use it. The gradient methods make use of the gradient $\nabla f(x)$ to determine the optimal search direction. Some gradient methods also use the Hessian H of the objective function $f(x)$. Conjugate-gradient methods use the gradient ∇f and an estimate of the Hessian H of the objective function $f(x)$.

Perpendicular search method

The perpendicular search method does not use gradient information, but makes use of the ordered set

$$S_1 = (d_0, d_1, \dots, d_{n-1})$$

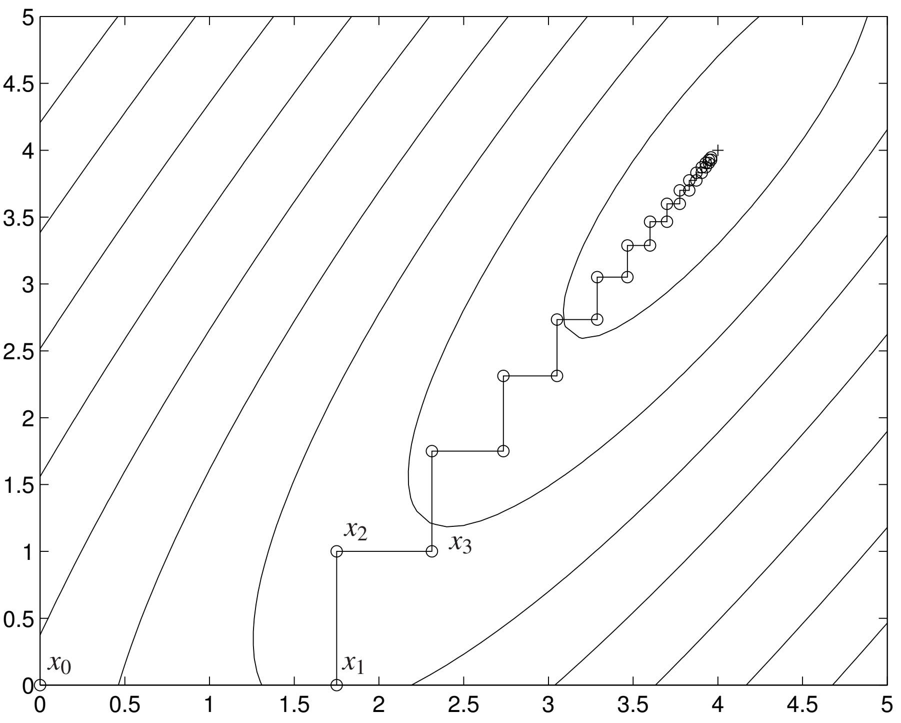


Figure 4.3: Perpendicular method.

of perpendicular basis vectors in $\mathbb{R}^n$ with

$$\begin{aligned} d_0 &= \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \end{bmatrix}^T \\ d_1 &= \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \end{bmatrix}^T \\ &\vdots \\ d_{n-1} &= \begin{bmatrix} 0 & 0 & 0 & \dots & 1 \end{bmatrix}^T \ . \end{aligned}$$

In each iteration a line search is done along direction d_k. If after n iterations the final solution x_k is not good enough, we perform a new cycle of n iterations with the same set of directions, and so on. So

$$S_2 = S_3 = \cdots = (d_0, d_1, \dots, d_{n-1}) \ .$$

Powell's perpendicular method

A disadvantage of perpendicular search is that the search directions cannot be adapted to the shape of the objective function f. If the axes of the objective function do not coincide with the parameter axes, a slow convergence can be expected. This disadvantage is avoided in Powell's perpendicular method. Powell has proposed search directions that are adapted to the objective function. Powell's perpendicular method uses subsequent cycles of n iteration steps with a different set of search directions for each cycle. It starts with the same ordered set

$$S_1 = (d_0, d_1, \dots, d_{n-1})$$

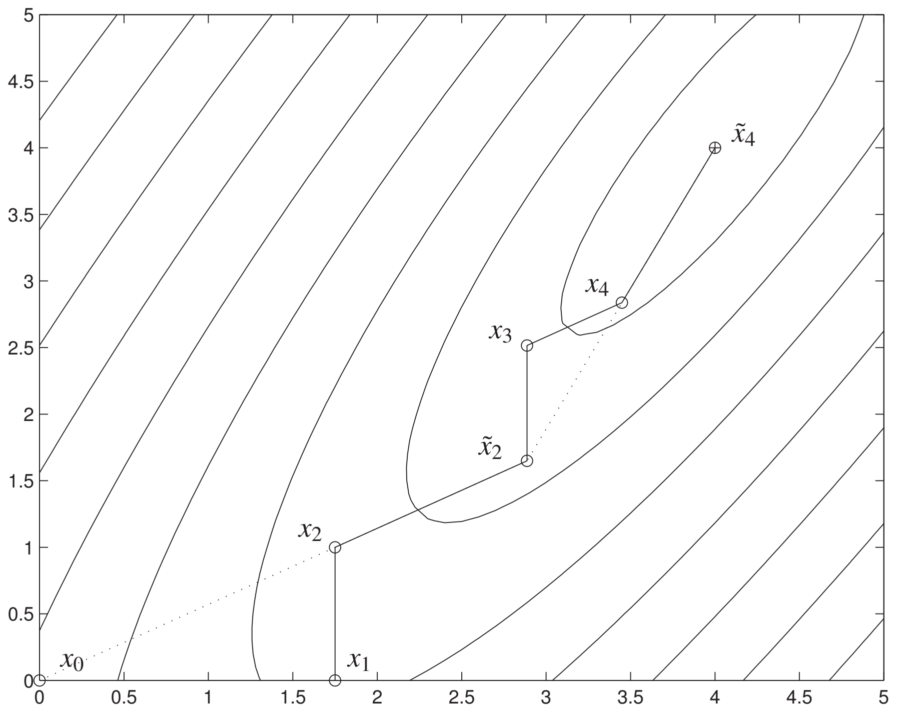


Figure 4.4: Powell's perpendicular method.

of perpendicular basis vectors in $\mathbb{R}^n$ as in the perpendicular search method. Consider the starting parameter vector $\tilde{x}_0 = x_0$ and the optimal parameter vector x_n obtained after n iteration steps. We now define a new search direction $d_n = x_n - \tilde{x}_0$, and we also search the minimum in the direction of d_n. This yields the new initial point $\tilde{x}_n$ for the next cycle of n iterations. The second cycle of n iterations uses a new set S_2 of search directions that is constructed from S_1 as follows: the first basis vector is thrown away, the remaining basis vectors are shifted one position to the left, and the new basis vector $d_n = x_n - \tilde{x}_0$ is added because it is in the direction of the largest decrease of the objective function. So the second ordered set of search directions becomes:

$$S_2 = (d_1, d_2, \ldots, d_{n-1}, \underbrace{x_n - \tilde{x}_0}_{d_n}) \ .$$

Now we perform n iterations starting from the point $\tilde{x}_n$, which yields the point x_{2n}. Next, we construct a new search direction $d_{n+1} = x_{2n} - \tilde{x}_n$, search for the optimum in this direction, and construct a new set S_3 of search directions for the next cycle of n iterations by shifting the basis vectors again and adding the new basis vector d_{n+1}:

$$S_3 = (d_2, d_3, \ldots, d_{n-1}, \underbrace{x_n - \tilde{x}_0}_{d_n}, \underbrace{x_{2n} - \tilde{x}_n}_{d_{n+1}}),$$

and so on.

Steepest descent direction:

The negative gradient is the direction in which $f(x)$ shows, locally, its largest decrease. It is called the *steepest descent direction* and is a reasonable choice for a search direction:

$$d_k = -\nabla f(x_k)$$

As we have seen in Section 4.1.1, the objective function f can be approximated locally by a second-order Taylor approximation and the minimum of this approximation is obtained by $x_{k+1} = x_k - H^{-1}(x_k)\nabla f(x_k)$. If the second-order approximation is not very precise, the error by making this move can become very large, and the algorithm may even not converge.
However, the difference between the current parameter vector x_k and the approximation of the local minimum x_{k+1} can be chosen as a search direction:

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

Of course, we can choose the Levenberg-Marquardt direction and the quasi-Newton search directions, as introduced in Sections 4.1.2 and 4.1.3:

Levenberg-Marquardt direction:

$$d_k = -\Big(\lambda I + H(x_k)\Big)^{-1}\nabla f(x_k)$$

Broyden-Fletcher-Goldfarb-Shanno direction:

$$d_k = -\hat{H}_k^{-1}\nabla f(x_k)$$

where

$$\begin{aligned}\hat{H}_k &= \hat{H}_{k-1} + \frac{q_k q_k^T}{q_k^T s_k} - \frac{\hat{H}_{k-1} s_k s_k^T \hat{H}_{k-1}^T}{s_k^T \hat{H}_{k-1} s_k}\\ s_k &= x_k - x_{k-1}\\ q_k &= \nabla f(x_k) - \nabla f(x_{k-1})\end{aligned}$$

Davidon-Fletcher-Powell direction:

$$d_k = -\hat{D}_k \nabla f(x_k)$$

where

$$\begin{aligned}\hat{D}_k &= \hat{D}_{k-1} + \frac{s_k s_k^T}{q_k^T s_k} - \frac{\hat{D}_{k-1} q_k q_k^T \hat{D}_{k-1}^T}{q_k^T \hat{D}_{k-1} q_k}\\ s_k &= x_k - x_{k-1}\\ q_k &= \nabla f(x_k) - \nabla f(x_{k-1})\end{aligned}$$

Fletcher-Reeves direction:

Another method for finding a proper search direction is the Fletcher-Reeves method, which is based on choosing the current gradient $\nabla f(x_k)$, updated by the last search direction d_{k-1}, multiplied by a factor μ_k:

$$d_k = -\nabla f(x_k) + \mu_k d_{k-1}$$

where

$$\mu_k = \frac{\nabla^T f(x_k)\nabla f(x_k)}{\nabla^T f(x_{k-1})\nabla f(x_{k-1})} \quad .$$

The Broyden-Fletcher-Goldfarb-Shanno, Davidon-Fletcher-Powell and Fletcher-Reeves search directions are conjugate-gradient directions. These conjugate-gradient search directions have the property that, if f is a quadratic objective function, and if $x \in \mathbb{R}^n$ (so we have an n-dimensional parameter space), we only need a line search in exactly n search directions to find the optimum of f.

Compared with Fletcher-Reeves, the Davidon-Fletcher-Powell method demonstrates a better convergence. Still, Fletcher-Reeves, or its modification Polak-Ribière (see, e.g., [37]), is preferred for many applications where the overhead of Davidon-Fletcher-Powell is too large. For example, in applications with large values of n (e.g., $n > 1000$), both the size of the matrix $\hat{D}_k$, which requires n^2 memory locations, and the calculation time can become too large; then Fletcher-Reeves is preferred.

A comparison of the direction determination and line search optimization methods, as discussed in this section, is given in [43].

4.3 Nelder-Mead method

In this section we present the Nelder-Mead method[2] [30], which is based on a geometrical shape known as a simplex. A simplex in $\mathbb{R}^n$ exists of $n+1$ points x_i, $i = 1, \ldots, n+1$, which do not lie on a hyperplane. For example, a triangle is a simplex in $\mathbb{R}^2$.

The Nelder-Mead method will be explained in $\mathbb{R}^2$; the extension to n dimensions is straightforward. Note that it is generally assumed that the Nelder-Mead method is less efficient than the methods considered above for problems with more than 3–4 variables. However, if the objective function is highly discontinuous, the Nelder-Mead method may be more robust.

Consider the minimization of the objective function $f(x)$. Let (x_0, x_1, x_2) be the vertices of a simplex in $\mathbb{R}^2$ and compute $f(x_0)$, $f(x_1)$ and $f(x_2)$. Consider the case that $f(x_0) > f(x_1)$ and $f(x_0) > f(x_2)$. We compute a new point x_3 by reflection of x_0 in the point

$$x_c = \frac{x_1 + x_2}{2}$$

which is in the middle of x_1 and x_2. The point x_3 is given by:

$$x_3 = x_c + d \qquad \text{where} \qquad d = x_c - x_0$$

The main idea is that the direction d will likely give a descent direction, and so probably $f(x_3) < f(x_0)$. Now (x_1, x_2, x_3) are the vertices of a new simplex. Again we determine which vertex has the highest value for the objective function and we reflect this vertex in the center of the remaining vertices.

[2]This method is sometimes also called the nonlinear simplex method. However, in order to avoid confusion with the simplex method for linear programming, we will not use this name.

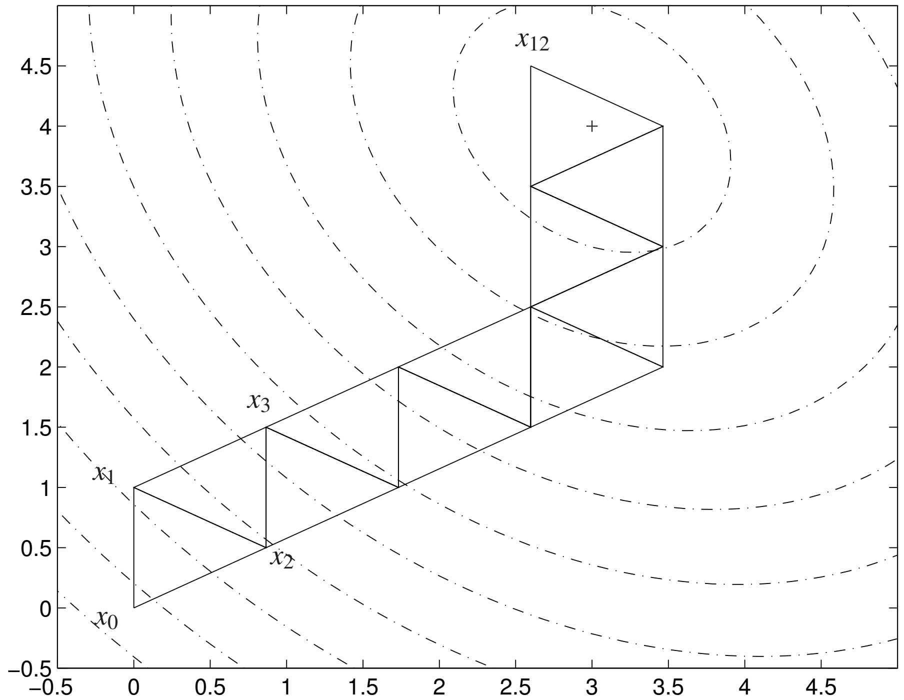


Figure 4.5: Reflection in the Nelder-Mead algorithm.

Repetition of this reflection procedure will lead to a sequence of decreasing values of $f(x_k)$ and when no reflection will give a smaller value of the objective function, we can terminate the procedure. The procedure of subsequent reflection is illustrated by Figure 4.5. Starting from point (x_0,x_1,x_2) we find the value x_{12} where another reflection will not lead to a smaller value of the objective function.

A disadvantage of the described method is that the volume of the simplex will not change and so if the starting vertex is very large, we may end far from the optimal value x^*. However, when we choose a small simplex, the number of steps to reach the optimal value x^* can be enormous. A big improvement can be made by scaling the reflection by a factor $\alpha \in \mathbb{R}^+$. The new point x_4 after reflection of x_1 now becomes:

$$x_4 = x_\mathrm{c} + \alpha\, d \qquad \text{where} \qquad d = x_\mathrm{c} - x_1 \ .$$

For $\alpha = 1$ we obtain the ordinary reflection;
for $\alpha > 1$ we obtain an expansion step, and the simplex will become larger;
for $\alpha < 1$ we obtain a contraction step, and the simplex will become smaller.
The variable size of the simplex can be used to obtain large steps if we are far from the optimum and small steps if we are close to the optimum.

Chapter 5

Constraints in Nonlinear Optimization

In this chapter we consider the optimization of a nonlinear objective function subject to equality or inequality constraints.

5.1 Equality constraints

In this section we consider the equality constrained minimization problem

$$\min_{x\in\mathbb{R}^n} f(x) \qquad \text{subject to } h(x) = 0 \ . \tag{5.1}$$

Sometimes this problem can be simplified by elimination of the equality constraint $h(x) = 0$, which leads to an unconstrained minimization problem. Unfortunately, explicit elimination of all the equality constraints is in general not always possible. Nevertheless, we can still simplify the problem by eliminating as many constraints $h_i(x) = 0$ as possible.

5.1.1 Linear equality constraints

Consider the following linear equality constrained minimization problem:

$$\min_{x\in\mathbb{R}^n} f(x) \qquad \text{subject to } Ax = b$$

where A is an $m\times n$ matrix, b is an $m\times 1$ vector, x is the $n\times 1$ parameter vector and where we assume that $n > m$. The elimination of the constraint $Ax = b$ can be done by defining a new parameter vector $\bar{x}\in\mathbb{R}^{n-m}$ and choosing an $(n-m)\times n$-matrix $\bar{A}$ and a vector x_0 such that $Ax_0 = b$ and $A\bar{A}^T = 0$. It is clear that for all $x = x_0 + \bar{A}^T\bar{x}$ there holds: $Ax = Ax_0 + (A\bar{A}^T)\bar{x} = b$. We obtain a new unconstrained optimization problem:

$$\min_{\bar{x}\in\mathbb{R}^{n-m}} f(x_0 + \bar{A}^T\bar{x})$$

The derivation of x_0 and $\bar{A}$ can be done as follows: Compute a singular value decomposition of A:

$$A = U \begin{bmatrix} \Sigma & 0 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} = U\Sigma V_1^T$$

where Σ is a diagonal matrix with nonnegative diagonal entries, and U and $[V_1\ V_2]$ are orthogonal matrices (See also Appendix C). Define $\bar{A} = V_2^T$ and $x_0 = V_1\Sigma^{-1}U^T b$. Then we find

$$A\bar{A}^T = U\Sigma V_1^T V_2 = 0\ , \ \text{because } \ V_1^T V_2 = 0$$

and

$$Ax_0 = U\Sigma V_1^T V_1 \Sigma^{-1} U^T b = b$$

because $V_1^T V_1 = I$ and $UU^T = I$.

5.1.2 Nonlinear equality constraints

Consider the equality constrained minimization problem when $h(x)$ is a nonlinear function of x. We can solve this problem using the Lagrange multiplier method. The equality constrained minimization problem is replaced by an unconstrained minimization problem:

$$\min_{x,\lambda} f(x) + \lambda^T h(x) \ , \tag{5.2}$$

where λ is a $p \times 1$ vector if there are p constraints $h_i(x) = 0\,,\ i = 1,\ldots,p$. The objective function of this unconstrained optimization problem has reached an extremum if the gradient is equal to zero:

$$\nabla_{x,\lambda}\left(f(x) + \lambda^T h(x)\right) = 0$$

or

$$\begin{aligned} \nabla_x f(x) + \nabla_x h(x)\,\lambda &= 0 \\ \text{and}\ \ \nabla_\lambda\left(\lambda^T h(x)\right) = h(x) &= 0\ \ . \end{aligned}$$

which are the Lagrange conditions from Chapter 1. We see that in the optimum (x^*, λ^*) the equality condition $h(x^*) = 0$ is satisfied and a variation of x away from x^* will either change the value of f or violate the equality condition $h(x) = 0$.
Hence, we can solve the *unconstrained* optimization problem (5.2) — using the methods of Chapter 4 — to obtain a minimum of the optimization problem with nonlinear equality constraints (5.1).

5.2 Inequality constraints

In this section we consider the inequality constrained minimization problem

$$\min_{x\in\mathbb{R}^n} f(x) \qquad \text{subject to } g(x) \leqslant 0$$

This problem can be solved by elimination, a gradient-projection method or by introducing penalty or barrier functions.

5.2.1 Elimination

Elimination of inequality constraints can be done if there exists a mapping $\Phi : \bar{x} \longrightarrow x$ such that

$$\{\, x \,|\, x = \Phi(\bar{x}),\, \bar{x} \in \mathbb{R}^m \,\} = \{\, x \,|\, x \in \mathbb{R}^n\,,\, g(x) \leqslant 0 \,\}$$

The constrained optimization of $f(x)$ over $x \in \mathbb{R}^n$ for $g(x) \leqslant 0$ now becomes an unconstrained optimization of $f(\Phi(\bar{x}))$ over $\bar{x} \in \mathbb{R}^m$.

Example 5.1 Consider the inequality constraint $x \geqslant 0$, $x \in \mathbb{R}$ and define the mapping $x = \bar{x}^2$. Noting that $\Phi(\bar{x}) = \bar{x}^2$ and $g(x) = -x$, we find

$$\{x \,|\, x = \bar{x}^2, \bar{x} \in \mathbb{R}\} = \{x \,|\, x \in \mathbb{R}, -x \leqslant 0\}$$

and so instead of the problem

$$\min_{x \in \mathbb{R}} f(x) \qquad \text{subject to } x \geqslant 0$$

we obtain the unconstrained problem

$$\min_{\bar{x} \in \mathbb{R}} f(\bar{x}^2) \ . \qquad \square$$

Example 5.2 Now we consider the constraint $2 \leqslant x \leqslant 4$, $x \in \mathbb{R}$, so $g(x) = \begin{bmatrix} x-4 \\ -x+2 \end{bmatrix} \leqslant 0$.

By introducing $\Phi(\bar{x}) = 3 + \cos(\bar{x})$ the inequality constrained problem

$$\min_{x \in \mathbb{R}} f(x) \qquad \text{subject to } 2 \leqslant x \leqslant 4$$

becomes an unconstrained problem:

$$\min_{\bar{x} \in \mathbb{R}} f(3 + \cos(\bar{x})) \ . \qquad \square$$

5.2.2 Linear inequality constraints

Gradient projection method

The gradient projection method can be applied in combination with a direction determination and line search optimization method. Consider the minimization problem

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

with linear inequality constraints:

$$Ax - b \leqslant 0$$

where $A \in \mathbb{R}^{m \times n}$ such that $\operatorname{rank}(A) = m$, and with the initial point x_k. Now we can distinguish between two cases. If we are in the interior of the feasible region or if we are on the boundary of the feasible region and the negative gradient $-\nabla f(x_k)$ points towards the feasible region, we use the negative gradient as search direction. If we are on the boundary of the feasible region and the negative gradient does not point towards the feasible region, we have to find another search direction. The main idea is to search for the direction where f decreases, but which is “tangent” to the boundary. This means we have to look at the active constraints, that means all constraints i for which $A_{i,\bullet} x_k = b_i$, where $A_{i,\bullet}$ is the ith row of A and b_i is the ith element of vector b. We construct the submatrix A_{act} and the subvector b_{act} consisting of respectively all rows $A_{i,\bullet}$ and all elements b_i that correspond to the active boundary constraints. So

$$A_{\text{act}} x_k = b_{\text{act}} \ .$$

We can find a projection matrix P that projects the negative gradient $-\nabla f(x_k)$ on the linear subspace that is tangent to the active constraints:

$$P = I - A_{\text{act}}^T (A_{\text{act}} A_{\text{act}}^T)^{-1} A_{\text{act}} \ .$$

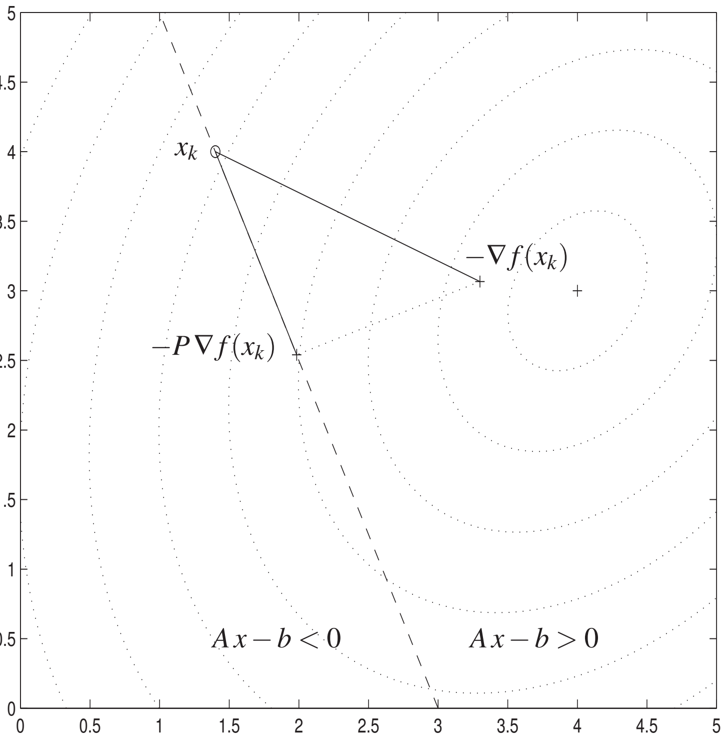


Figure 5.1: Gradient-projection method.

The new search direction is defined by

$$d_k = -P\nabla f(x_k) \quad .$$

We obtain the one-dimensional minimization problem:

$$\min_{s\in\mathbb{R}} f(x_k + d_k s) \qquad \text{subject to } A\,(x_k + d_k s) - b \leqslant 0 \quad .$$

5.2.3 Nonlinear inequality constraints

Nonlinear inequality constraints usually make the optimization problem more difficult, because the boundary of the allowed region is no longer a hyperplane, on which the gradient can be projected. An easy way to tackle the nonlinear inequality constraint problem is to incorporate the constraint function into the objective function, using a penalty or a barrier function.

The basic idea of the penalty method and the barrier method is the same. Introduce a feasibility function f_{feas} defined by

$$\begin{aligned} f_{\text{feas}}(x) &= 0 \qquad \text{if } \max_i g_i(x) \leqslant 0 \quad (\text{or equivalently: } g(x) \leqslant 0) \\ f_{\text{feas}}(x) &= \infty \qquad \text{if } \max_i g_i(x) > 0 \quad (\text{or equivalently: } g(x) \nleqslant 0) \ , \end{aligned}$$

and add this function to the original objective function, so that we obtain the new minimization problem

$$\min_x \big(f(x) + f_{\text{feas}}(x)\big) \quad .$$

A minimization of this new objective function will always find a minimum in the allowed region where $g(x) \leqslant 0$.

A problem with this feasibility function is that it is not smooth and so the use of a gradient method to find the optimum will fail. There are two ways to approximate the feasibility function, namely with a penalty function or with a barrier function.

Penalty function method

A penalty function f_{pen} has the property

$$\begin{aligned} f_{\text{pen}}(x) &= 0 \qquad \text{if } \max_i g_i(x) \leqslant 0 \quad \text{(or equivalently: } g(x) \leqslant 0) \\ f_{\text{pen}}(x) &\gg 0 \qquad \text{if } \max_i g_i(x) > 0 \quad \text{(or equivalently: } g(x) \nleqslant 0) \ . \end{aligned}$$

Examples of penalty functions are:

$$\begin{aligned} f_{\text{pen}}(x) &= \beta \sum_{i=1}^{m} \max\Big(0, g_i(x)\Big) \ , & \beta \gg 1 \\ f_{\text{pen}}(x) &= \beta \sum_{i=1}^{m} \max\Big(0, g_i(x)\Big)^2 \ , & \beta \gg 1 \\ f_{\text{pen}}(x) &= \max_i \max\,(0, e^{\beta\, g_i(x)} - 1)^2 \ , & \beta \gg 1 \ . \end{aligned}$$

If g_i is smooth, then the second and third choice have the advantage that the derivatives are continuous in x so that gradient-based optimization techniques can be used to minimize the new objective function.

Example 5.3 Consider the minimization problem

$$\min_{x \in \mathbb{R}} \,(x-5)^2 \qquad \text{subject to } x^2 - 1 \leqslant 0 \ .$$

If we choose the following penalty function

$$f_{\text{pen}}(x) = \max\,(0, e^{5\,g(x)} - 1)^2 = \max(0, e^{5(x^2-1)} - 1)^2$$

the minimum is found $x^* \approx 1.026$. The true optimum $x^* = 1$, which is on the boundary, can only be approximated by increasing β. For $\beta \longrightarrow \infty$ we will obtain the ideal feasibility function again. For big β the computation of the gradient will become numerically difficult. □

A problem with penalty functions is that if the true optimum is on the boundary, the computed optimum might be infeasible. This problem can be avoided by using barrier functions.

Barrier function method

A barrier function f_{bar} has the property

$$\begin{aligned} f_{\text{bar}}(x) &\approx 0 \quad \text{for } \max_i g_i(x) \ll 0 \\ f_{\text{bar}}(x) &\to \infty \quad \text{for } \max_i g_i(x) \uparrow 0 \ , \end{aligned}$$

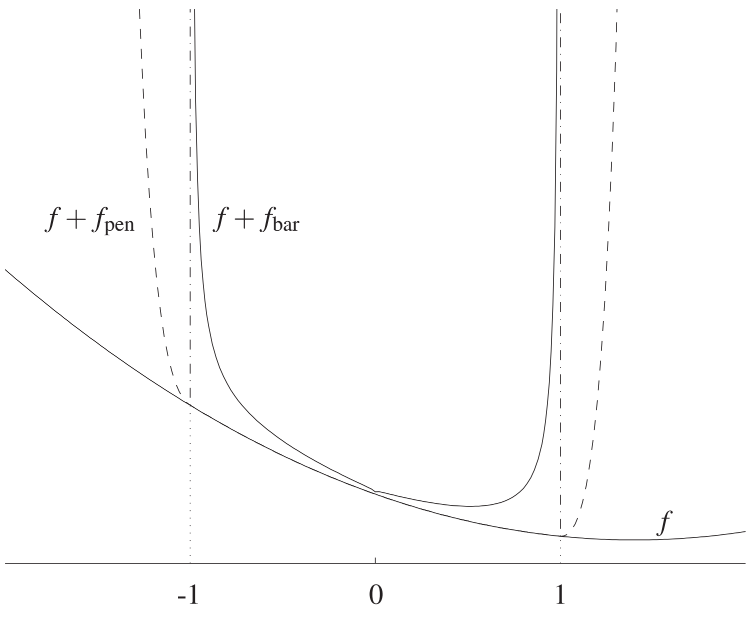


Figure 5.2: Penalty and barrier functions. The dash-dotted line corresponds to $f + f_{\text{feas}}$

and is usually undefined for $\max_i g_i(x) \geqslant 0$. Examples of barrier functions are:

$$f_{\text{bar}}(x) = -\frac{1}{\beta}\sum_{i=1}^{m} \log\left(-g_i(x)\right) , \qquad \beta > 1$$

$$f_{\text{bar}}(x) = \sum_{i=1}^{m} -\frac{1}{\beta g_i(x)} , \qquad \beta > 1$$

$$f_{\text{bar}}(x) = -\frac{1}{\beta} \log\left(-\max_i g_i(x)\right) , \qquad \beta > 1$$

where log is the natural logarithm.

Example 5.4 Again, consider the minimization problem

$$\min_{x \in \mathbb{R}} (x-5)^2 \qquad \text{subject to } x^2 - 1 \leqslant 0 \ .$$

If we choose the following barrier function

$$f_{\text{bar}}(x) = -\frac{1}{100\, g(x)} = \frac{1}{100\,(1-x^2)}$$

the minimum is found $x^* \approx 0.975$. The true optimum $x^* = 1$, which is on the boundary, can only be approximated by increasing β. For $\beta \longrightarrow \infty$ we will obtain the ideal feasibility function again. For big β the computation of the gradient will become numerically difficult. □

A disadvantage of the barrier function is that *during* the iteration care should be taken that the new point x_{k+1} does not enter the infeasible region, because the barrier function is not defined there. The barrier function is sometimes called internal penalty function; the penalty function, as defined in this section, is then called the external penalty function.

Sequential quadratic programming

In this section the sequential quadratic programming (SQP) algorithm [2], a state-of-the art algorithm, is discussed to solve the following optimization problem with inequality constraints:

$$\min_x f(x) \qquad \text{subject to } \; g(x) \leqslant 0 \;\; .$$

In Chapter 4 we have considered a direction determination and line search optimization algorithm with quasi-Newton directions for solving an unconstrained optimization problem. The sequential quadratic programming algorithm is based on a constrained version of this algorithm.

We start by introducing the Lagrange function

$$L(x,\lambda) = f(x) + \lambda^T g(x) \;\; .$$

We make a second-order approximation of this function at x_k:

$$L(x,\lambda) \approx L(x_k,\lambda) + \nabla_x^T L(x_k,\lambda)\,(x-x_k) + \frac{1}{2}(x-x_k)^T H_L(x_k,\lambda)\,(x-x_k)$$

where H_L is the Hessian of L with respect to x. We make a linear approximation of the inequality constraints:

$$g(x) \approx g(x_k) + \nabla^T g(x_k)\,(x-x_k) \;\; .$$

From Chapter 4 we know that the computation of the Hessian matrix $H_L(x_k,\lambda)$ may be time-consuming and the inversion is sometimes bothersome. By replacing $H_L(x_k,\lambda)$ by the Broyden-Fletcher-Goldfarb-Shanno approximation $\hat{H}_k$, we avoid numerical problems and speed up the optimization. In order to determine the next estimate of the optimum of our original optimization problem with inequality constraints, we define $d = x - x_k$ and we solve the following quadratic programming (QP) problem[1]:

$$\min_d \; \frac{1}{2} d^T \hat{H}_k d + \nabla^T f(x_k) d$$

$$\text{subject to } \; g(x_k) + \nabla^T g(x_k) d \leqslant 0 \;\; .$$

The optimum d_k of this QP subproblem can be used to determine a new estimate: $x_{k+1} = x_k + d_k$. However, we would better use it as a new search direction and perform the line search

$$s_k = \arg\min_s \; \psi(x_k + s d_k)$$

where ψ is a merit function that incorporates both the objective function and the constraints using, e.g., a penalty function or a barrier function. This yields the next estimate

$$x_{k+1} = x_k + s_k d_k \;\; .$$

The final algorithm becomes:

Step 1: Compute gradient of $L(x_k,\lambda)$ and approximate the Hessian by $\hat{H}_k$.

[1]This formulation, i.e., containing the (approximate) Hessian of L and the gradient of f (instead of L), is the one most often found in literature.

Step 2: Find the search direction d_k by solving the following QP problem:

$$\begin{aligned}&\min_d \ \frac{1}{2} d^T \hat{H}_k d + \nabla^T f(x_k) d \\ &\text{subject to } \ g(x_k) + \nabla^T g(x_k) d \leqslant 0 \ \ .\end{aligned}$$

Set $\Delta_k = \lambda^* - \lambda_k$ where λ^* is the optimal Lagrange multiplier of the above QP problem.

Step 3: Perform a line optimization

$$s_k = \arg\min_s \ \psi(x_k + s\, d_k)$$

where ψ is a merit function (e.g., the sum of the objective function and a penalty function).

Step 4: Define the new estimate

$$\begin{aligned}x_{k+1} &= x_k + s_k\, d_k \\ \lambda_{k+1} &= \lambda_k + s_k \Delta_k\end{aligned}$$

Step 5: If optimum is not found, then go to step 1.

We see that the SQP algorithm performs the same steps as a direction determination and line search optimization algorithm with quasi-Newton directions. The differences are that we replace a minimization of the objective function $f(x)$ by a minimization of the Lagrange function $L(x,\lambda)$. Furthermore, the search direction is not found by solving an unconstrained quadratic problem (leading to $d_k = -\hat{H}_k^{-1} \nabla f(x_k)$), but it is found by solving a (constrained) quadratic programming problem.

Chapter 6

Convex Optimization

This chapter is about convex optimization, i.e., we consider optimization problems for which both the objective function and the constraint functions are convex. In Chapter 1 the concept of convexity was already discussed. The use of the convexity property to solve optimization problems is attractive, because algorithms for convex optimization are easily implemented and provide simple stopping criteria that guarantee that the optimum is found up to a pre-specified accuracy.

In Section 6.1 some basic convex functions are reviewed and properties that preserve convexity will be discussed. Two specific convex problems are discussed in Sections 6.2 and 6.3. A special set of convex objective functions is introduced, namely that of norm evaluation of affine functions. Next a recent development in control engineering is studied, where controller design problems are solved by translating design specifications into a set of linear matrix inequalities. In Section 6.4 three convex optimization algorithms will be discussed, the cutting-plane algorithm, the ellipsoid algorithm and the interior-point method. We conclude this chapter with a control engineering example in Section 6.5.

6.1 Convex functions

In Chapter 1 the concept of convexity was already discussed. In this section some basic convex functions are given and operations are discussed that preserve convexity.

Here are some basic convex functions:

$$
\begin{array}{ll}
f(x) = \alpha x & \text{with } \alpha \in \mathbb{R} \\
f(x) = \alpha x^{2n} & \text{with } \alpha \in \mathbb{R}^+,\ n \in \mathbb{N}_0 \\
f(x) = x^p & \text{for } x \in \mathbb{R}^+ \text{ with } p \geqslant 1 \\
f(x) = e^x & \\
f(x) = -\log(x) & \text{for } x \in \mathbb{R}_0^+ \\
f(x) = \cosh(x) = \dfrac{e^x + e^{-x}}{2} & \\
f(x) = h\Big(\beta_0 + \sum\limits_{i=1}^{n} \beta_i x_i\Big) & \text{with } \beta_i \in \mathbb{R}^m, \text{ and } h(y) \text{ convex for all } y \in \mathbb{R}^m\,.
\end{array}
$$

A norm function $\|\cdot\|$ defined on $\mathbb{R}^n$ is a function that satisfies the following properties:

1. $\|u\| \geqslant 0$,

2. $\|u\| = 0 \Leftrightarrow u = 0$,
3. $\|au\| = |a|\|u\| \quad \forall a \in \mathbb{R}$,
4. $\|u+v\| \leqslant \|u\| + \|v\|$,

for any $u, v \in \mathbb{R}^n$. The p-norm (with $p \geqslant 1$) of a vector $x \in \mathbb{R}^n$ is defined as follows:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} .$$

It is easy to verify that this function indeed satisfies the properties of a norm for $p \geqslant 1$.

Using the definitions of convexity and of a norm function it is easy to prove that every norm function is convex. Here, it is important to note that this means that every norm function is convex *in its own argument*, e.g., the absolute value function $|x|$ is convex in its argument x, but the function $|\sin x|$ is *not* convex in x.

The following operations preserve convexity:

- **Positive-weighted sums:**
  Let the functions f_i be convex for $i = 1, \ldots, n$. Then the function f defined by

  $$f(x) = \sum_{i=1}^{n} \alpha_i f_i(x) \ , \ \ \alpha_i \in \mathbb{R}^+$$

  will be convex. An extension to infinite sums can be given. Let $\mathscr{A}$ be a set such that the function $g(\cdot, y)$ is convex for any fixed $y \in \mathscr{A}$. Then the function f defined by

  $$f(x) = \int_{\mathscr{A}} w(y) g(x, y) \, dy$$

  is convex in x, provided $w(y) \geqslant 0$ for each $y \in \mathscr{A}$.

- **Pointwise maximum:**
  Let the functions f_i be convex for $i = 1, \ldots, n$. Then the function f defined by

  $$f(x) = \max_{i=1,\ldots,n} f_i(x)$$

  will be convex. The pointwise maximum property extends to the positive supremum over an infinite set of convex functions. Let $\mathscr{A}$ be a set such that the function $g(\cdot, y)$ is convex for any fixed $y \in \mathscr{A}$. Then the function f defined by

  $$f(x) = \max_{y \in \mathscr{A}} g(x, y)$$

  is convex.

Example 6.1
The function f defined by $f(x) = \max_{i=1,\ldots,n} (a_i^T x + b_i)$ is convex.
The function f defined by $f(x) = \max_{y \in \mathscr{S}} \|x - y\|_2$ is convex if $\mathscr{S} \subset \mathbb{R}^n$. $\square$

- **Composition:**
  Let f, g and h be two-times differentiable functions. And let

  $$f(x) = h(g(x))$$

  Then f is convex if either g is convex and h is convex and non-decreasing, or $-g$ is convex and h is convex and non-increasing.

  Example 6.2
  The function f defined by $f(x) = \exp g(x)$ is convex if g is convex.
  The function f defined by $f(x) = 1/g(x)$ is convex if $g(x)$ positive for all x and $-g$ is convex[1].
  The function f defined by $f(x) = \Big(g(x)\Big)^p$, $p \geqslant 1$, is convex if g is convex and nonnegative.
  The function f defined by $f(x) = -\log g(x)$ is convex if g is positive and $-g$ is convex[1]. □

6.2 Convex problems: Norm evaluation of affine functions

We now consider some specific criteria defined by the 1-norm, the squared 2-norm and the ∞-norm of a discrete-time signal, which is affine in the parameter vector $x = (x_1 \ldots, x_N)$. This type of criterion plays an important role in multi-criteria controller design, as will be discussed in the second part of these lecture notes. The criteria[2] are given by

$$\begin{aligned}
f_1(x) &= \| F_0(n) + F_1^T(n)x \|_1 = \sum_n \big|F_0(n) + F_1^T(n)x\big| \\
f_2(x) &= \| F_0(n) + F_1^T(n)x \|_2^2 = \sum_n \big(F_0(n) + F_1^T(n)x\big)^2 \\
f_\infty(x) &= \| F_0(n) + F_1^T(n)x \|_\infty = \max_n \big|F_0(n) + F_1^T(n)x\big| \ ,
\end{aligned}$$

where F_0 is a scalar discrete-time signal, and F_1 is a vector-valued signal containing N scalar discrete-time signals. An important property of the above criteria is that they are convex in the parameter vector x. This can easily be proved by combining the properties of preservation of convexity by positive weighted sums, pointwise maximum and the property that the norm function $\|\cdot\|$ is convex.

For the 1-norm the subgradient becomes:

$$\nabla f_1(x) = \sum_n F_1(n)\,\mathrm{sign}\Big(F_0(n) + F_1^T(n)x \Big)$$

For the squared 2-norm the subgradient becomes:

$$\nabla f_2(x) = \sum_n 2\,F_1(n)\,\Big(F_0(n) + F_1^T(n)x\Big)$$

For the ∞-norm the subgradient becomes:

$$\nabla f_\infty(x) = F_1(n_x)\,\mathrm{sign}\Big(F_0(n_x) + F_1^T(n_x)x \Big)$$

[1] Note that it is not required that the domain of g is equal to $\mathbb{R}$ (in which case only trivial functions would satisfy the requirement); the domain of g can also be a (convex) subset of $\mathbb{R}$.

[2] These definitions are taken from the book of Kwakernaak and Sivan [24]. See also Section 13.4.1.

where

$$n_x = \arg\max_n |\, F_0(n) + F_1^T(n)x \,| \ .$$

We will now show that $\nabla f_\infty(x)$ is indeed a subgradient of $f_\infty(x)$. We have to prove that for any $y \in \mathbb{R}$ we have

$$f_\infty(x) \ \geqslant \ f_\infty(y) \ + \ \nabla^T f_\infty(y)\,(x-y) \quad \text{for all } \ x \in \mathbb{R}\,.$$

For ease of notation we define:

$$n_x = \arg\max_n |\, F_0(n) + F_1^T(n)\,x \,|$$

$$n_y = \arg\max_n |\, F_0(n) + F_1^T(n)\,y \,|$$

$$\beta_y = \text{sign}\Big(\, F_0(n_y) + F_1^T(n_y)\,y \,\Big)$$

We have

$$\begin{aligned}
f_\infty(x) &= \|\, F_0(n) + F_1^T(n)\,x \,\|_\infty \\
&= \max_n |\, F_0(n) + F_1^T(n)\,x \,| \\
&= |\, F_0(n_x) + F_1^T(n_x)\,x \,| \\
&\geqslant |\, F_0(n_y) + F_1^T(n_y)\,x \,| \\
&\geqslant \beta_y \Big(\, F_0(n_y) + F_1^T(n_y)\,x \Big) \\
&\geqslant \beta_y F_0(n_y) + \beta_y F_1^T(n_y)\,y + \beta_y F_1^T(n_y)\,(x-y) \\
&\geqslant |\, F_0(n_y) + F_1^T(n_y)\,y \,| + F_1^T(n_y)\,\beta_y\,(x-y) \\
&\geqslant \max_n |\, F_0(n) + F_1^T(n)\,y \,| + F_1^T(n_y)\,\text{sign}\Big(\, F_0(n_y) + F_1^T(n_y) y \,\Big)(x-y) \\
&\geqslant f_\infty(y) \ + \ \nabla^T f_\infty(y)\,(x-y)
\end{aligned}$$

where we have used the following properties:

$$\max_n \left|F_0(n) + F_1^T(n)x\right| = \left|F_0(n_x) + F_1^T(n_x)x\right| \geqslant \left|F_0(n_y) + F_1^T(n_y)x\right|$$

$$\text{and} \qquad \left|F_0(n) + F_1^T(n)y\right| \geqslant \beta\big(F_0(n) + F_1^T(n)y\big) \qquad \text{for } \beta = \pm 1\,.$$

6.3 Convex problems: Linear matrix inequalities

The concept of linear matrix inequalities has caused a dramatic change in control theory. Many classical control problems and various until recently unsolvable control problems can be recast as a convex optimization problem, with a convex objective function and a system of linear matrix inequalities as constraints. Such a convex optimization problem can be solved using very reliable and fast algorithms, such as the interior-point method (which will be presented in Section 6.4.3).

Let $A \in \mathbb{R}^{n \times n}$. Recall that we say that A is positive definite, denoted by $A > 0$, if $x^T A x > 0$ for all $x \in \mathbb{R}^n$ with $x \neq 0$. In this section we will also use the following properties (cf. Section 1.3): if $A, B > 0$ and if X is a non-singular matrix, then $A^T > 0$, $A + B > 0$, and $XAX^T > 0$.

6.3.1 Linear Matrix Inequalities (LMIs)

Consider the linear matrix expression

$$F(x) = F_0 + \sum_{i=1}^{m} x_i F_i \tag{6.1}$$

where $x \in \mathbb{R}^m$ is the variable and the symmetric matrices $F_i = F_i^T \in \mathbb{R}^{n\times n}$, $i = 0,\dots,m$ are given. A Linear Matrix Inequality (LMI) has the form

$$F(x) > 0 \ . \tag{6.2}$$

Here are some properties of LMIs:

- **Convexity:**
  The set $\{\, x \,|\, F(x) > 0 \,\}$ is convex.
- **Multiple LMIs:**
  Multiple LMIs can be expressed as a single LMI, since

  $$F^{(1)}(x) > 0\ ,\ F^{(2)}(x) > 0\ ,\dots,\ F^{(p)}(x) > 0$$

  is equivalent to:

  $$\begin{bmatrix} F^{(1)}(x) & 0 & \cdots & 0 \\ 0 & F^{(2)}(x) & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & F^{(p)}(x) \end{bmatrix} > 0$$
- **Schur complement:**
  Consider the following LMI:

  $$\begin{bmatrix} Q(x) & S(x) \\ S^T(x) & R(x) \end{bmatrix} > 0$$

  where $Q(x) = Q^T(x)$, $R(x) = R^T(x)$ and $S(x)$ depend affinely on x. This is equivalent to

  $$R(x) > 0\ ,\ Q(x) - S(x)R^{-1}(x)S^T(x) > 0$$

  or

  $$Q(x) > 0\ ,\ R(x) - S^T(x)Q^{-1}(x)S(x) > 0\ \ .$$

6.3.2 LMIs in system and control engineering

In fact, the field of application of LMIs in system and control engineering is enormous and more and more engineering problems are solved using LMIs. Some examples of applications are stability theory, model and controller reduction, robust control, model predictive control and system identification.

- **Exponential stability**
  Probably the most elementary LMI is related to the exponential stability. The differential equation:

  $$\frac{d}{dt}x(t) = A\,x(t)$$

is exponentially stable (i.e., all trajectories go to the equilibrium point 0) if and only if there exists a positive definite matrix P such that

$$A^T P + PA < 0 \ .$$

Note that we may assume without loss of generality that P is symmetric[3]. It is easy to verify that in that case the conditions $P > 0$ and $A^T P + PA < 0$ can be recast as an LMI.

- **Stabilizing state-feedback**
Consider the differential equation:

$$\frac{d}{dt}x(t) = Ax(t) + Bu(t)$$

where $x(t)$ is the state and $u(t)$ is the input. When we apply a constant-gain state feedback $u(t) = Kx(t)$, we obtain:

$$\frac{d}{dt}x(t) = Ax(t) + BKx(t)$$

and so the corresponding Lyapunov equation for stability becomes:

$$(A^T + K^T B^T)P + P(A + BK) < 0 \ , \quad P > 0 \ .$$

Since P is non-singular and symmetric, this is equivalent to

$$P^{-1}\big((A^T + K^T B^T)P + P(A + BK)\big)P^{-1} < 0 \ .$$

If we define $P = X^{-1}$, this leads to

$$X(A^T + K^T B^T) + (A + BK)X < 0 \ , \quad X > 0$$

By choosing $K = YX^{-1}$ we obtain the condition:

$$XA^T + Y^T B^T + AX + BY < 0 \ , \quad X > 0$$

which is an LMI in X and Y. So any X and Y satisfying the above LMI result in a stabilizing state-feedback $K = YX^{-1}$.

- **Quadratic objective**
Consider the system

$$\frac{d}{dt}x(t) = Ax(t) + Bu(t)$$

with $x(0) = x_0$ given. Now we consider the problem of finding an optimal constant-gain state feedback $u(t) = Kx(t)$ to minimize the quadratic objective

$$\min_K J(K)$$

with

$$\begin{aligned} J(K) &= \int_0^\infty (x^T Qx + u^T Ru)\, dt \\ &= \int_0^\infty x^T (Q + K^T RK)x\, dt \end{aligned}$$

[3]Indeed if $P > 0$ and $A^T P + PA < 0$, then $P^T > 0$ and $A^T P^T + P^T A < 0$. So if we define $Q = P + P^T$, then Q is symmetric, $Q > 0$ and $A^T Q + QA < 0$.

where $Q, R > 0$.
Now we show that this problem can be translated into a minimization problem with LMI constraints. Consider the quadratic function $V(z) = z^T P z$ ($P > 0$) that satisfies

$$\frac{d}{dt}V(x) < -x^T(Q + K^T R K)x \tag{6.3}$$

for every trajectory. If the state feedback is stabilizing — which will be the case if we follow the procedure given below, — we have $\lim_{t\to\infty} x(t) = 0$ and thus also $\lim_{t\to\infty} V(x(t)) = 0$. Therefore, integrating (6.3) from 0 to ∞ yields

$$V(x_0) > J(K) \ .$$

Hence, $x_0^T P x_0$ is an upper bound on $J(K)$.
Since

$$\begin{aligned}\frac{d}{dt}V(x) = \frac{d}{dt}(x^T P x) &= \frac{dx^T}{dt}Px + x^T P \frac{dx}{dt} \\ &= x^T\big((A+BK)^T P + P(A+BK)\big)x \ ,\end{aligned}$$

condition (6.3) is equivalent to

$$(A+BK)^T P + P(A+BK) + Q + K^T R K < 0 \ .$$

By setting $P = \gamma X^{-1}$ and $K = Y X^{-1}$ we obtain the following condition:

$$XA^T + Y^T B^T + AX + BY + \gamma^{-1} XQX + \gamma^{-1} Y^T R Y < 0 \tag{6.4}$$

which can be rewritten (using the Schur complement transformation) as

$$\begin{bmatrix} -(XA^T + Y^T B^T + AX + BY) & XQ^{1/2} & Y^T R^{1/2} \\ Q^{1/2} X & \gamma I & 0 \\ R^{1/2} Y & 0 & \gamma I \end{bmatrix} > 0\,,\ X > 0\,,\ \gamma > 0 \tag{6.5}$$

So any γ, X and Y satisfying the above LMI gives an upper bound

$$V(x_0) = \gamma x_0^T X^{-1} x_0 > J \ .$$

If we add the normalization constraint $x_0^T X^{-1} x_0 < 1$, or equivalently

$$\begin{bmatrix} 1 & x_0^T \\ x_0 & X \end{bmatrix} > 0 \ ,$$

then minimizing γ subject to LMI-constraints (6.5) leads to a minimization of the upper bound of the quadratic objective.
Note that the LMI constraint (6.5) looks more complicated than expression (6.4). However, (6.5) is linear in X, Y and γ. The interior point algorithm, which will be discussed in Section 6.4.3, can solve convex optimization problems with many LMI constraints (up to a 1000 constraints) in a fast and robust way.

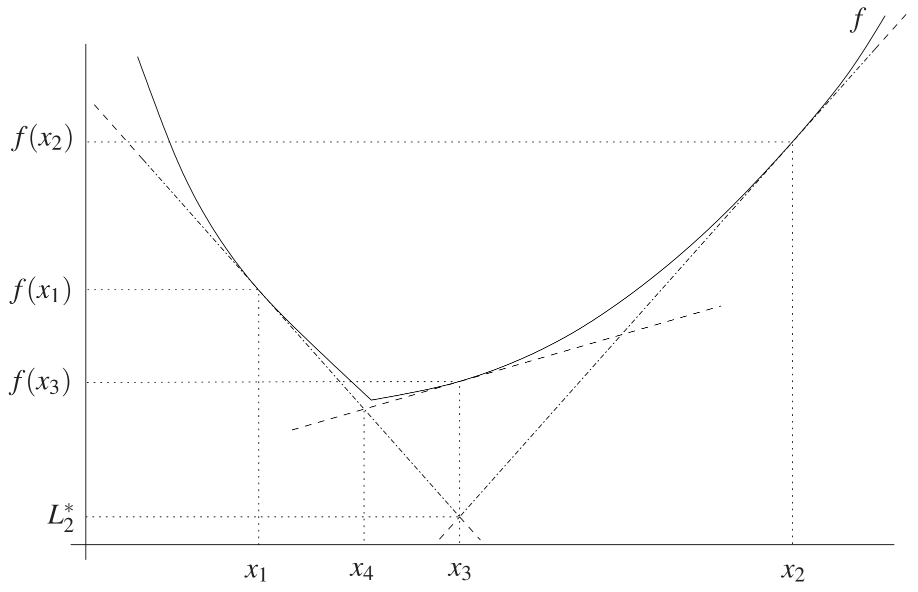


Figure 6.1: Illustration of the cutting-plane algorithm for the one-dimensional case. The initial points are x_1 and x_2. The dashed lines correspond to the tangent lines in the points $(x_k, f(x_k))$, i.e., the functions on the right-hand side of (6.6). The dash-dotted piecewise-linear function corresponds to the right-hand side of (6.7), and the point $x^* = x_3$ minimizes this function. In the next step the tangent line in $(x_3, f(x_3))$ will be included, which will result in x_4.

6.4 Convex optimization techniques

In this section we discuss some algorithms to solve convex optimizations problems, i.e., problems of the following form:

$$\begin{aligned}
&\min_x f(x) \\
&\text{s.t. } g(x) \leqslant 0 \\
&\text{with } f(x) \text{ and } g(x) \text{ convex functions.}
\end{aligned}$$

Note that in general a constraint of the form $h(x) = 0$ is *not* convex if h is convex, but that the constraint $h(x) = 0$ is convex only if h is an affine function[4].

6.4.1 Cutting-plane algorithm

In this section we discuss the cutting plane algorithm [3, 12].

Let the objective function f be convex for all x that belong to the convex set $\mathscr{C}$. Suppose that for the k parameter vectors $x_1, \ldots, x_k$ we have calculated the values of the objective function $f(x_1), \ldots, f(x_k)$ and the subgradients $\nabla f(x_1), \ldots, \nabla f(x_k)$. Now we know:

$$f(x) \geqslant f(x_i) + \nabla^T f(x_i)(x - x_i) \qquad \text{for } i = 1, 2, \ldots, k. \tag{6.6}$$

[4]This can be verified by considering the equivalent constraint $h(x) \leqslant 0$, $-h(x) \leqslant 0$, which is convex if both h and $-h$ are convex, i.e., if h is an affine function.

So for all x we have

$$f(x) \geqslant \max_{i=1,\ldots,k}\Big(f(x_i) + \nabla^T f(x_i)(x-x_i)\Big) \tag{6.7}$$

and so for x^* there will hold

$$f(x^*) \geqslant \min_x \max_{i=1,\ldots,k}\Big(f(x_i) + \nabla^T f(x_i)(x-x_i)\Big) =: L_k^* \ .$$

It is easy to verify (see also Proposition 8.3) that L_k^* can be determined by solving the following linear programming problem:

$$\begin{aligned} &L_k^* = \min_{x,L_k} L_k \\ &\text{s.t. } L_k \geqslant f(x_i) + \nabla^T f(x_i)(x-x_i) \qquad \text{for } i=1,\ldots,k \end{aligned}$$

If we define the following matrices and vectors:

$$c = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad w_k = \begin{bmatrix} x \\ L_k \end{bmatrix},$$

$$A = \begin{bmatrix} \nabla^T f(x_1) & -1 \\ \vdots & \vdots \\ \nabla^T f(x_k) & -1 \end{bmatrix}, \quad b = \begin{bmatrix} \nabla^T f(x_1)x_1 - f(x_1) \\ \vdots \\ \nabla^T f(x_k)x_k - f(x_k) \end{bmatrix},$$

we can rewrite the problem more compactly as

$$\min_{w_k} c^T w_k \qquad \text{subject to } A\,w_k \leqslant b \ .$$

If $w_k^* \in \mathbb{R}^{n+1}$ is an optimizer of this problem, then $L_k^* = \begin{bmatrix} 0 & 1 \end{bmatrix} w_k^*$.

If we define $U_k^* = \min_{i=1,\ldots,k} f(x_i)$, we obtain the following bounds:

$$U_k^* \geqslant f(x^*) \geqslant L_k^* \ .$$

If the interval size $U_k^* - L_k^*$ is sufficiently small, we can take x_k^* as a reasonable estimate for x^*. If not, we define $x_{k+1} = x_k^*$ and add the row and the element that correspond to x_{k+1} to the matrix A and the vector b. We can calculate a new upper bound U_{k+1}^* and lower bound L_{k+1}^*, where there will hold

$$U_k^* \geqslant U_{k+1}^* \geqslant f(x^*) \geqslant L_{k+1}^* \geqslant L_k^* \ .$$

We can iterate until the stopping criterion is satisfied (e.g., if $U_k^* - L_k^*$ is sufficiently small).

Handling constraints

Now we consider the cutting-plane algorithm with constraints, related to the following optimization problem:

$$\min_x f(x) \qquad \text{subject to } g(x) \leqslant 0$$

where f and g are convex functions. For the sake of simplicity we assume that g is a scalar function. However, the extension to multiple constraints is straightforward.

Recall that for the subgradient of a convex function (g) we have

$$g(x) \geqslant g(x_i) + \nabla^T g(x_i)(x - x_i) \qquad \text{for } i = 1, \ldots, k \ .$$

This means that

$$\{x \,|\, g(x) \leqslant 0\} \subseteq \left\{x \,|\, g(x_i) + \nabla^T g(x_i)\,(x - x_i) \leqslant 0\right\} \quad \text{for } i = 1, \ldots, k \ .$$

So we get an *outer* approximation of the feasible set, if we introduce the extra linear constraints

$$\nabla^T g(x_i)\, x \leqslant \nabla^T g(x_i)\, x_i - g(x_i) \quad \text{for } i = 1, \ldots, k$$

and include them into the linear programming procedure by defining:

$$c = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad w_k = \begin{bmatrix} x \\ L_k \end{bmatrix},$$

$$A = \begin{bmatrix} \nabla^T f(x_1) & -1 \\ \nabla^T g(x_1) & 0 \\ \vdots & \vdots \\ \nabla^T f(x_k) & -1 \\ \nabla^T g(x_k) & 0 \end{bmatrix}, \quad b = \begin{bmatrix} \nabla^T f(x_1) x_1 - f(x_1) \\ \nabla^T g(x_1) x_1 - g(x_1) \\ \vdots \\ \nabla^T f(x_k) x_k - f(x_k) \\ \nabla^T g(x_k) x_k - g(x_k) \end{bmatrix},$$

and solving the linear programming problem

$$\min_{w_k} c^T w_k \qquad \text{subject to } A\, w_k \leqslant b \ .$$

Again we find the optimum $x_k^* = \begin{bmatrix} I & 0 \end{bmatrix} w_k^*$, a lower bound $L_k^* = \begin{bmatrix} 0 & 1 \end{bmatrix} w_k^*$ and an upper bound $U_k^* = \min\{f(x_i^*) \,|\, i = 1, \ldots, k,\ g(x_i^*) \leqslant 0\}$ such that

$$L_k^* \leqslant f(x^*) \leqslant U_k^* \ .$$

In general, the x_k^*'s may be infeasible (since we have used an outer approximation of the feasible set). However, it can be shown [3, 12] that $\lim\limits_{k \to \infty} g(x_k^*) \leqslant 0$.

Remark 6.3 For functions that are nearly convex in some area we can still use the cutting-plane algorithm. However, we may expect some errors in running the algorithm if we find $U_i < L_i$. In that case we have to choose a new set of starting vectors $x_1, \ldots, x_k$ in the neighborhood of the last-found x_i, and then we can restart the optimization. ◇

6.4.2 Ellipsoid algorithm

In this section we discuss a second convex optimization algorithm, namely the ellipsoid algorithm [3, 20].

The ellipsoid algorithm is based on the fact that the direction of the subgradient divides the search space into two halves. To make this clear we consider the one-dimensional case (see Figure 6.2).

Let $x \in \mathbb{R}$ and suppose we know that the minimum is in the interval

$$E_0 = \{x \,|\, x_0 - A_0 \leqslant x \leqslant x_0 + A_0\}$$

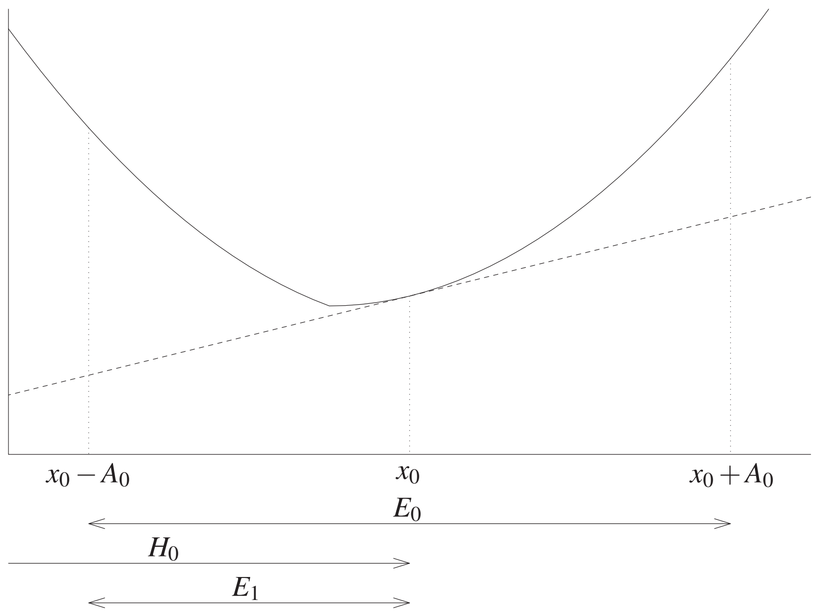


Figure 6.2: The ellipsoid method for the one-dimensional case.

with $x_0, A_0 \in \mathbb{R}$. We can calculate a subgradient $\nabla f(x_0)$ in x_0. Recall that the subgradient inequality states that for a convex function f we have

$$f(x) \geqslant f(x_0) + \nabla^T f(x_0)(x - x_0) \qquad \text{for all } x \ .$$

As a consequence, the point x^* for which f reaches a minimum, will be in the half-line H_0 defined by

$$H_0 = \{x \mid \nabla^T f(x_0)(x - x_0) \leqslant 0\} \ .$$

We now know that $H_0 \cap E_0$ will contain the minimizer x^*. We can construct a new interval

$$E_1 = \{x \mid x_1 - A_1 \leqslant x \leqslant x_1 + A_1\}$$

such that $E_1 = H_0 \cap E_0$ and such that the size of E_1 is smaller than the size of E_0, by choosing

$$A_1 = A_0/2 \qquad \text{and} \qquad x_1 = x_0 - A_1 \operatorname{sign}\Big(\nabla f(x_0)\Big)$$

Now we consider the algorithm for the multi-dimensional case. Suppose we know that the minimum x^* is in the ellipsoid

$$E_0 = \{x \mid (x - x_0)^T A_0^{-1}(x - x_0) \leqslant 1\}$$

where $A_0 \in \mathbb{R}^{n \times n}$ is non-singular and positive definite. We can calculate a subgradient $\nabla f(x_0)$ in x_0. Since f is convex, x^* will be in the half-space H_0 defined by

$$H_0 = \{x \mid \nabla^T f(x_0)\,(x - x_0) \leqslant 0\} \ .$$

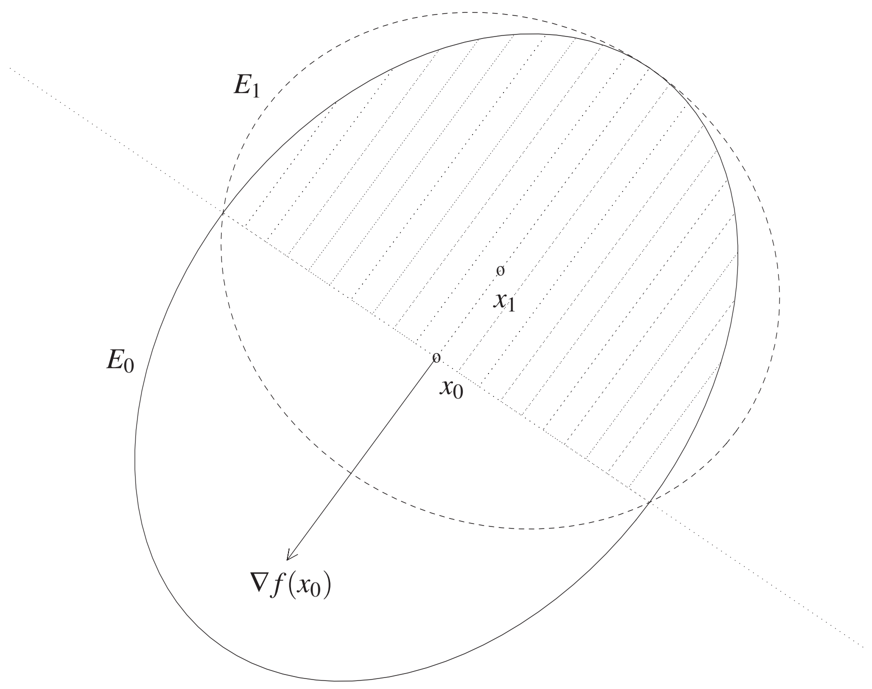


Figure 6.3: The ellipsoid method for the two-dimensional case.

We now know that the half-ellipsoid $H_0 \cap E_0$ will contain the minimizer x^*. We can construct a new ellipsoid

$$E_1 = \{x \mid (x-x_1)^T A_1^{-1}(x-x_1) \leqslant 1\}$$

such that $E_1 \supseteq H_0 \cap E_0$ and such that the volume of E_1 is less than the volume of E_0.
This can be repeated in an iterative procedure. Now suppose that after k iterations we know that the minimum x^* is in the ellipsoid

$$E_k = \{x \mid (x-x_k)^T A_k^{-1}(x-x_k) \leqslant 1\} \ .$$

Let $\nabla f(x_k)$ be the subgradient of f in x_k. If we compute

$$x_{k+1} = x_k - \frac{1}{n+1} \frac{A_k \nabla f(x_k)}{\sqrt{\nabla^T f(x_k) A_k \nabla f(x_k)}} \tag{6.8}$$

$$A_{k+1} = \frac{n^2}{n^2-1}\left(A_k - \frac{2}{n+1} \frac{A_k \nabla f(x_k) \nabla^T f(x_k) A_k^T}{\nabla^T f(x_k) A_k \nabla f(x_k)} \right) \ , \tag{6.9}$$

then we obtain the $(k+1)$th ellipsoid:

$$E_{k+1} = \{x \mid (x-x_{k+1})^T A_{k+1}^{-1}(x-x_{k+1}) \leqslant 1\} \ .$$

The volume of the consecutive ellipsoids will decrease to 0 for $k \to \infty$. Furthermore, we have

$$f(x_k) - f(x^*) \leqslant \sqrt{\nabla^T f(x_k) A_k \nabla f(x_k)} \ .$$

Handling constraints

The ellipsoid algorithm in the constrained case is very similar to the unconstrained case. For the sake of simplicity we assume that g is a scalar function.

We start with an ellipsoid

$$E_0 = \{x \mid (x - x_0)^T A_0^{-1} (x - x_0) \leqslant 1\}$$

that contains the optimal, and therefore feasible, solution x^*. Suppose that at a certain point we have obtained x_k. Now the next ellipsoid is computed as follows:

- If $g(x_k) \leqslant 0$ or, equivalently, if x_k is feasible, we compute the next ellipsoid in the same way as for the unconstrained case. So the parameters x_{k+1} and A_{k+1} of the $(k+1)$th ellipsoid are defined by (6.8)–(6.9).
  In this case the iteration step is called an "objective iteration", i.e., we discard points that cannot minimize the objective function.

- If $g(x_k) > 0$ or, equivalently, if x_k is infeasible, we use a modified computation to obtain the next ellipsoid, by replacing $\nabla f(x_k)$ by $\nabla g(x_k)$ in the expression for x_{k+1} and A_{k+1}. If $g(x_k) - \sqrt{\nabla^T g(x_k) A_k \nabla g(x_k)} > 0$, there is no feasible solution, because the feasible set is empty and we stop the algorithm. Otherwise, there is a feasible solution and we compute x_{k+1} and A_{k+1} as follows:

$$x_{k+1} = x_k - \frac{1}{n+1} \frac{A_k \nabla g(x_k)}{\sqrt{\nabla^T g(x_k) A_k \nabla g(x_k)}}$$

$$A_{k+1} = \frac{n^2}{n^2 - 1} \left(A_k - \frac{2}{n+1} \frac{A_k \nabla g(x_k) \nabla^T g(x_k) A_k^T}{\nabla^T g(x_k) A_k \nabla g(x_k)} \right) \ ,$$

  This iteration step is called a "constraint iteration", i.e., we discard infeasible points.

Remark 6.4 The ellipsoid algorithm is not only applicable to convex optimization problems, but also to quasi-convex optimization problems, which makes it more powerful than the cutting-plane algorithm. $\diamond$

6.4.3 Interior-point algorithms

In this section we discuss a third convex optimization algorithm, namely the interior-point method [32]. This method solves the constrained convex optimization problem

$$f^* = f(x^*) = \min_x f(x) \quad \text{subject to} \quad g(x) \leqslant 0$$

and is based on the barrier function method, discussed in section 5.2.3. Consider the non-empty strictly feasible set

$$\mathcal{G} = \{ x \mid g_i(x) < 0, \ i = 1, \ldots, m \}$$

and define a barrier function ϕ as follows:

$$\phi(x) = \begin{cases} -\sum_{i=1}^{m} \log(-g_i(x)) & x \in \mathcal{G} \\ \infty & \text{otherwise} \end{cases}$$

Note that ϕ is convex on $\mathcal{G}$ and, as x approaches the boundary of $\mathcal{G}$, the function ϕ will go to infinity. Now consider the unconstrained optimization problem

$$\min_x \big(t\, f(x) + \phi(x)\big)$$

for some $t \geqslant 0$. Then the value $x^*(t)$ that minimizes this function is denoted by

$$x^*(t) = \arg\min_x \big(t\, f(x) + \phi(x)\big) \quad . \tag{6.10}$$

It is clear that the curve $x^*(t)$, which is called the central path, is always located inside the set $\mathcal{G}$. The variable weight t gives the relative weight between the objective function f and the barrier function ϕ, which is a measure for the constraint violation. Intuition suggests that $x^*(t)$ will converge to the optimal value x^* of the original problem as $t \longrightarrow \infty$. Note that in a minimum $x^*(t)$ the gradient of $\psi(x,t) = t\, f(x) + \phi(x)$ with respect to x is zero, so:

$$\nabla_x\, \psi(x^*, t) = t \nabla_x f(x^*(t)) + \sum_{i=1}^{m} \frac{1}{-g_i(x^*(t))} \nabla_x g_i(x^*(t)) = 0 \quad .$$

We can rewrite this as

$$\nabla_x f(x^*(t)) + \sum_{i=1}^{m} \lambda_i^*(t) \nabla_x g_i(x^*(t)) = 0$$

where

$$\lambda_i^*(t) = \frac{1}{-g_i(x^*(t))\, t} \quad . \tag{6.11}$$

Then from the above we can derive that at the optimum the following conditions hold:

$$g(x^*(t)) \leqslant 0 \tag{6.12}$$

$$\lambda_i^*(t) \geqslant 0 \tag{6.13}$$

$$\nabla_x f(x^*(t)) + \sum_{i=1}^{m} \lambda_i^*(t) \nabla_x g_i(x^*(t)) = 0 \tag{6.14}$$

$$\lambda_i^*(t) g_i(x^*(t)) = -1/t \tag{6.15}$$

which proves that for $t \to \infty$ the right-hand side of the last condition will go to zero and so the Kuhn-Tucker conditions (see Section 1.5.1) will be satisfied. This means that $x^*(t) \to x^*$ and $\lambda^*(t) \to \lambda^*$ for $t \to \infty$.

From the above we can see that the central path can be regarded as a continuous deformation of the Kuhn-Tucker conditions.

Now define the dual function [37]:

$$d(\lambda) = \min_x \left(f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) \right) \quad \text{for } \lambda \in \mathbb{R}^m .$$

A property of this dual function is that [37]:

$$d(\lambda) \leqslant f^* \qquad \text{for any } \lambda \geqslant 0$$

and so we have

$$\begin{aligned}
f^* &\geqslant d(\lambda^*(t)) \\
&\geqslant \min_x \left(f(x) + \sum_{i=1}^m \lambda_i^*(t) g_i(x) \right) \\
&\geqslant f(x^*(t)) + \sum_{i=1}^m \lambda_i^*(t) g_i(x^*(t)) && \text{(since } x^*(t) \text{ also minimizes the function} \\
& && F \text{ defined by } F(x) = f(x) + \sum_{i=1}^m \lambda_i^*(t) g_i(x) \\
& && \text{since } \nabla_x F(x^*(t)) = 0 \text{ by (6.14))} \\
&= f(x^*(t)) - m/t && \text{(by (6.11))}
\end{aligned}$$

and thus

$$f(x^*(t)) \geqslant f^* \geqslant f(x^*(t)) - m/t$$

It is clear that m/t is an upper bound for the distance between the computed $f(x^*(t))$ and the optimum f^*. If we want a desired accuracy $\varepsilon > 0$, defined by

$$|\, f^* - f(x^*(t))\,| \leqslant \varepsilon$$

then choosing $t = m/\varepsilon$ gives us one unconstrained minimization problem (6.10), with a solution $x^*(m/\varepsilon)$ that satisfies:

$$|\, f^* - f(x^*(m/\varepsilon))\,| \leqslant \varepsilon \ .$$

To solve the unconstrained minimization problem the methods discussed in Chapter 4 can be used.

The approach presented above works, but can be very slow. A better way is to increase t in steps to the desired value m/ε and use each intermediate solution as a starting value for the next iteration. This algorithm is called the sequential unconstrained minimization technique (SUMT) and is described by the following steps

```
Given:  $x \in \mathcal{G}$, $t > 0$ and a tolerance $\varepsilon$
Step 1: Compute $x^*(t)$ starting from $x$: $x^*(t) = \arg\min_x \big(t\, f(x) + \phi(x)\big)$.
Step 2: Set $x = x^*(t)$.
Step 3: If $m/t \leqslant \varepsilon$, return $x$ and stop.
Step 4: Increase $t$ and goto step 1.
```

This algorithm generates a sequence of points on the central path and solves the constrained optimization problem via a sequence of unconstrained minimizations (often quasi-Newton methods). A simple update rule for t is used, so $t(k+1) = \eta\, t(k)$, where η is typically between 10 and 100.

The steps 1–4 are called the outer iteration and step 1 involves an inner iteration (e.g., the Newton steps). A trade-off has to be made: a smaller η means fewer inner iterations to compute x_{k+1} from x_k, but more outer iterations.

In Nesterov and Nemirovsky [32] an upper bound is derived on the number of iterations that are needed for an optimization using the interior-point method. Also tuning rules for the choice of η are given in their book.

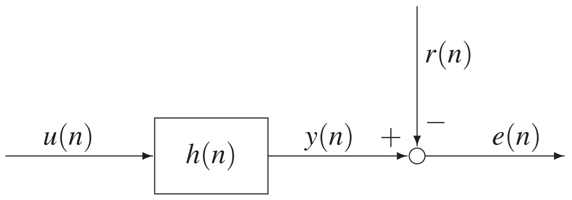


Figure 6.4: Control problem for Example 6.6.

Remark 6.5 The ellipsoid algorithm and the interior-point algorithm introduced in this chapter can also be used for linear and quadratic minimization problems with a positive (semi-)definite objective function, since these problems are also convex. In addition, for convex quadratic minimization problems the cutting-plane algorithm can also be used. However, for small to medium-sized linear and quadratic programming problems it is better to use (variants of) the methods discussed in Chapter 2 (simplex algorithm) and Chapter 3 (modified simplex algorithm) respectively. $\diamond$

6.5 Controller design example

In this section an optimization problem arising from control theory will be elaborated.

Example 6.6 Consider an initially-at-rest discrete-time linear time-invariant system (see Figure 6.4) with input signal u, output signal y and impulse response h given by

$$h(n) = \begin{cases} 2^{-n} & \text{for } 0 \leqslant n \leqslant 5,\ n \in \mathbb{N} \\ 0 & \text{otherwise} \end{cases} .$$

Assume that $u(n) = 0$ and $y(n) = 0$ for $n < 0$.

We like the output y to follow a given trajectory r defined by $r(n) = 1$ in the time range $\{0, 1, 2, \ldots, 10\}$. We can do that by making a good choice for the input signal u. We introduce an extra difficulty in the given problem by demanding $u(n) = 0$ for $n \in \{6, 7, \ldots, 10\}$. This means that we can only choose $u(n) \neq 0$ for $n \in \{0, 1, \ldots, 5\}$. The parameter vector for minimization is thus given by

$$x = \begin{bmatrix} u(0) \\ u(1) \\ \vdots \\ u(5) \end{bmatrix} .$$

We know (see [24]) that

$$y(n) = \sum_{m=0}^{5} h(n-m)\, u(m) = F_1^T(n)\, x$$

where

$$F_1^T(n) = \begin{bmatrix} h(n) & h(n-1) & \ldots & h(n-5) \end{bmatrix} .$$

By defining $F_0(n) = -r(n)$, we obtain the error signal

$$e(n) = y(n) - r(n) = F_0(n) + F_1^T(n)\, x \ .$$

$f(x)$	# iterations	evaluation $f_1(x^*)$	$f_2(x^*)$	$f_\infty(x^*)$
1-norm	12	4.083	3.480	0.979
squared 2-norm	8	4.141	3.157	0.959
∞-norm	7	8.650	7.567	0.922

Table 6.1: Optimization results for the three objective functions of example 6.6.

We shall consider three different norms of the error signal, namely the 1-norm, the squared 2-norm and the ∞-norm. This leads to the following objective functions:

$$f_1(x) = \| e(n) \|_1 = \sum_{n=0}^{10} | F_0(n) + F_1^T(n) x |$$

$$f_2(x) = \| e(n) \|_2^2 = \sum_{n=0}^{10} (F_0(n) + F_1^T(n) x)^2$$

$$f_\infty(x) = \| e(n) \|_\infty = \max_{n=0,1,\ldots,10} | F_0(n) + F_1^T(n) x | \ .$$

From Section 6.2 we know that these objective functions are convex, and that subgradients for the objective functions can be derived.

We do the optimization with $x_0 = 0$ using the cutting-plane algorithm with stopping criterion $U_k^* - L_k^* \leqslant 10^{-3}$. We obtain the results given in Table 6.1. The first column gives the objective function we have chosen, and the second column gives the number of iterations that are needed to satisfy the stopping criterion. The last three columns give the function values of f_1, f_2 and f_∞ in the optimum x^*.
The responses $y(n)$ for the optimal values of $x = [u(0) \ldots u(10)]^T$ are plotted in Figure 6.5.
Special attention should be paid to the ∞-norm case. We observe that the response is optimal in the sense that the maximum deviation between $r(n)$ and $y(n)$ is indeed minimized. This maximum deviation is reached for $n = 10$. This means that in this case all efforts are aimed at making the deviation $|r(10) - y(10)|$ minimal. As a result, the response for $n \in \{0,1,\ldots,5\}$ becomes very bad; this is because making $y(n) = r(n)$ for $n \in \{0,1,\ldots,5\}$ does not decrease the value of f_∞. For this example it becomes clear that we have to make sure that the chosen objective function $f(x)$ reflects our idea of optimality. □

Remark 6.7 The convex optimization for the squared 2-norm can be rewritten as a quadratic optimization problem by defining:

$$E = \begin{bmatrix} e(0) \\ e(1) \\ \vdots \\ e(10) \end{bmatrix} = \begin{bmatrix} F_0(0) + F_1^T(0) x \\ F_0(1) + F_1^T(1) x \\ \vdots \\ F_0(10) + F_1^T(10) x \end{bmatrix} = H_0 + H_1 x$$

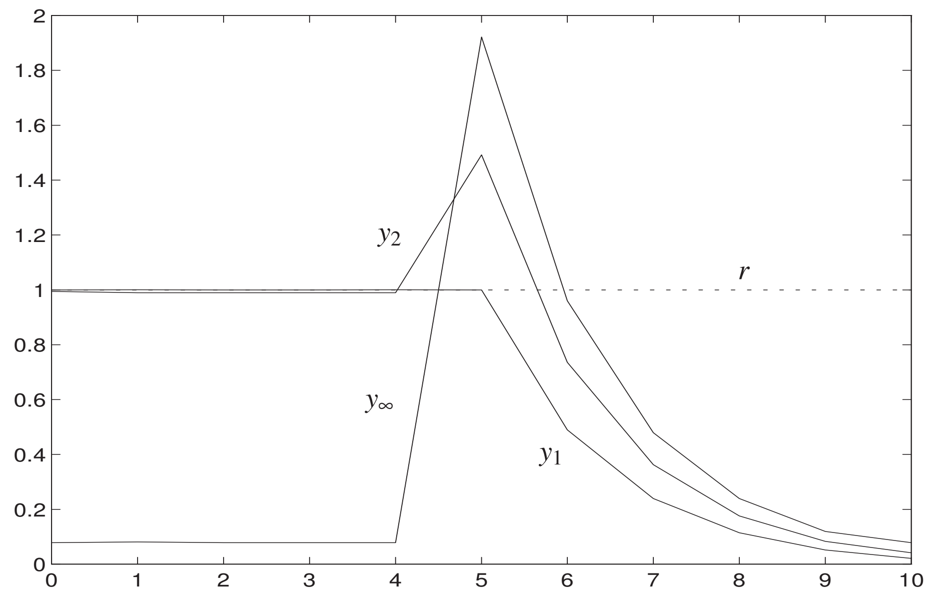


Figure 6.5: The reference signal r (dotted line) and various optimal output signals y (solid lines) of example 6.6 as a function of the time step n.

where

$$H_0 = \begin{bmatrix} F_0(0) \\ F_0(1) \\ \vdots \\ F_0(10) \end{bmatrix} \qquad H_1 = \begin{bmatrix} F_1^T(0) \\ F_1^T(1) \\ \vdots \\ F_1^T(10) \end{bmatrix}$$

The squared 2-norm of the error signal is given by

$$\begin{aligned} f_2(x) &= E^T E = x^T H_1^T H_1 x + x^T H_1^T H_0 + H_0^T H_1 x + H_0^T H_0 \\ &= \frac{1}{2} x^T H x + c^T x + d \end{aligned}$$

where

$$H = 2H_1^T H_1 \qquad c = 2H_1^T H_0 \qquad d = H_0^T H_0 \ .$$

Note that $x^T H_1^T H_0 = H_0^T H_1 x$ and that we can drop the term d, because this term does not influence the position of the minimum. Now we have an optimization problem with the objective function

$$\frac{1}{2} x^T H x + c^T x \ ,$$

which is an unconstrained quadratic optimization problem. $\diamond$

Exercises

Exercise 6.1 Are the following functions convex or not? Why?

1. $f : \mathbb{R} \to \mathbb{R} : x \mapsto (x^2+1)^2$
2. $f : \mathbb{R} \to \mathbb{R} : x \mapsto (x^2-3x)^2$
3. $f : \mathbb{R} \to \mathbb{R} : x \mapsto 2^x$
4. $f : \mathbb{R} \to \mathbb{R} : x \mapsto \left(\frac{1}{2}\right)^x$
5. $f : \mathbb{R} \setminus \{0\} \to \mathbb{R} : x \mapsto \frac{1}{x}$
6. $f : [1,+\infty) \to \mathbb{R} : x \mapsto \frac{1}{x}$
7. $f : \mathbb{R}^2 \to \mathbb{R} : (x,y) \mapsto \cosh(x^2+y^2)$

Exercise 6.2 On page 60 it is stated that if P is symmetric then the conditions $P > 0$ and $A^T P + PA < 0$ can be recast as an LMI. Prove this statement.
Hint: Write P as a linear combination of symmetric basis matrices, each having only one (diagonal) entry or two (off-diagonal) entries equal to 1, the other entries being equal to 0.

Chapter 7

Global Optimization

7.1 Local and global minima

Consider a general minimization problem with objective function f and feasible set $\mathscr{G}$. We say that a point $x^* \in \mathscr{G}$ is a *local minimum* if there exists a neighborhood[1] $\mathscr{N}$ of x^* such that for each point $x \in \mathscr{N} \cap \mathscr{G}$ we have $f(x) \geqslant f(x^*)$. The point x^* is a *global minimum* if for each point $x \in \mathscr{G}$ we have $f(x) \geqslant f(x^*)$.

For some of the minimization problems discussed in the previous chapters, every local minimum will also be global. This holds, e.g., for linear programming problems, quadratic programming problems (provided that the matrix H of the objective function is positive definite), convex optimization problems, and unconstrained problems with a unimodal objective function. However, for general nonlinear non-convex minimization problems, we usually have many local minima and not every local minimum will be a global minimum. The minimization methods discussed in Chapters 4 and 5 only return a local minimum. Therefore, in general we cannot be sure that the returned solution is indeed the global minimum. This implies that in general global optimization has to be used since in general there always is a possibility that there is more than one local optimum.

It can be shown that the problem of finding the global minimum is a computationally hard problem, in the sense that trying to obtain the exact global optimum may require an excessive amount of computation time (See also Chapter 11). Therefore, in practice we will usually use a method that — although convergence to the global optimum cannot be guaranteed — will yield "good" solutions on the average. There exist several methods that can be used to this end such as

- Random search
- Multi-start local optimization
- Simulated annealing
- Genetic algorithms
- Randomized algorithms
- ...

In the next sections we will discuss random search, multi-start local optimization, genetic algorithms, and simulated annealing. For more information on randomized algorithms the reader is referred to [23].

[1] A neighborhood of a point x^* could be defined as an open ball with center x^* and radius $\varepsilon > 0$.

7.2 Random search

The most elementary method to determine a global minimum is the random search method. In this method we select a large set of points that are, e.g., uniformly distributed over the feasible set. We then evaluate the values of the objective function and we select the point that yields the lowest function value. In general we have to consider a large amount of points to obtain a reasonable approximation of the global optimum.

7.3 Multi-start local optimization

Multi-start local optimization can be considered as a combination between random search and local optimization. In this method we select several starting points in the feasible set (using, e.g., a uniform distribution) and for each starting point we run a local minimization method. From the set of returned solutions we afterwards select the one that yields the lowest value for the objective function.

Although this method is widely used in practice, it is not always very efficient. However, if based on physical insights or on additional information about the problem, or by using approximations we already have a good idea of the region where the global optimum will be situated, we can select our initial points in this region and then the process will be much more efficient. This topic will also be discussed in more detail in Part II of these lecture notes.

7.4 Simulated annealing

In this section we consider the following minimization problem:

$$\min_x f(x) \quad \text{s.t. } x \in \mathcal{G}$$

where $\mathcal{G}$ is the set of feasible solutions.

In simulated annealing [14] we mimic the annealing of metal; the objective function to be minimized then corresponds to the energy. Molten metal is a liquid and then the particles are arranged randomly. As we start to cool the metal, the movement of particles is restricted. For slow cooling we end up with a crystal-like structure, which corresponds to a minimum energy configuration. Fast cooling on the other hand results in a glass-like structure or a structure with defects in the crystal lattice. This is a metastable, locally optimal structure. We could say that a descent algorithm like the local minimization methods of Chapters 4 and 5 corresponds to fast cooling. The approach used in the simulated annealing method will correspond to a slow cooling, thus resulting in a more global optimization.

The basic simulated annealing algorithm can be formulated as follows:

- **Initialization**:
  select an initial point $x(0)$, an initial "temperature" $T(0) > 0$, a constant temperature interval length $m \in \mathbb{N}_0$, and a cooling factor $\alpha \in (0,1)$
  set $k = 0$

- **Iteration**:
  generate a random point $x_{\text{n}} \in \mathcal{G}$ in the neighborhood of $x(k)$
  define $\delta = f(x_{\text{n}}) - f(x(k))$
  if $\delta < 0$ then

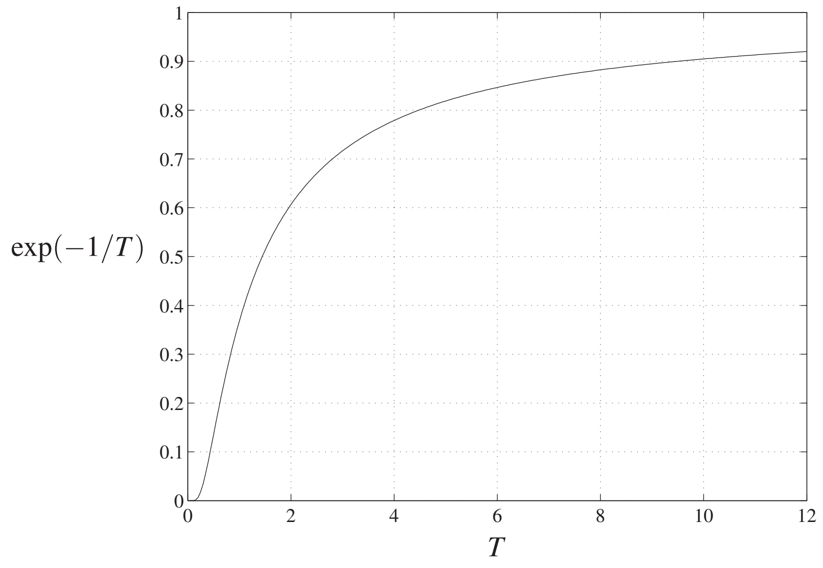


Figure 7.1: The graph of the function $\exp(-\delta/T)$ for $\delta = 1$.

```
        $x(k+1) \leftarrow x_\text{n}$                     *(down-hill move)*
    else
        select a random number $r \in (0,1)$
        if $r < \exp(-\delta/T)$ then
            $x(k+1) \leftarrow x_\text{n}$                 *(up-hill move)*
        else
            $x(k+1) \leftarrow x(k)$
    if $m$ divides $k$ then
        $T(k+1) \leftarrow \alpha T(k)$                    *(cooling step)*
    else
        $T(k+1) \leftarrow T(k)$
    $k \leftarrow k+1$
```

- **Stop** if the stopping criterion is satisfied (e.g., the maximum number of iterations has been reached).

Note that an important difference with the methods of Chapters 4 and 5 is that sometimes an up-hill move (which increases f) is accepted. However, small increases are more likely to be accepted than large increases as the iteration goes on: If the temperature T is high, then most up-hill moves are accepted, since then the probability that the random number r is less than $\exp(-\delta/T)$ is high (cf. Figure 7.1). If the temperature T is low, then most up-hill moves are rejected, since then the probability that $r < \exp(-\delta/T)$ is low.

Some characteristics of the simulated annealing algorithm are:

- Sometimes a move which increases the objective function is accepted.
- The algorithm can escape from local minima.
- The algorithm uses probabilistic transition rules.

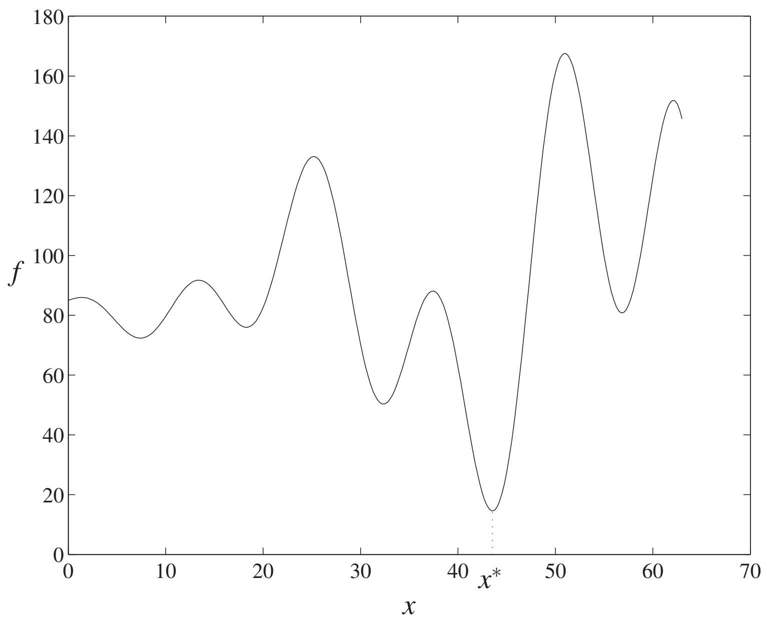


Figure 7.2: The objective function of the minimization problem of Example 7.1. The global minimum is reached at $x = x^* \approx 43.493$.

- The algorithm can also be used for problems with discrete parameters (See also Chapter 11).
- Only the function values of the objective function are required; the gradient and the Hessian are not used.

Example 7.1 Consider the following optimization problem:

$$\begin{aligned} &\min_x f(x) \;=\; \min_x \;\; 85 + (1-x)\sin\frac{x}{5} + x\cos\frac{x}{2} \\ &\text{s.t. } \; x \in [0, 63] \end{aligned}$$

The evolution of the objective function f is represented in Figure 7.2. If we start, e.g., in the point $x_0 = 10$ then a local optimization algorithm will always end up in the local minimum at $x = 7.3$. However, the simulated annealing method can jump over the hills located around $x = 13.2$, $x = 25.2$ or $x = 37.4$ and end up in the global minimum or a local minimum with a lower value of the objective function. The evolution of the current iteration point, its function value and the temperature for a typical run of the simulated annealing algorithm is represented in Figure 7.3. □

7.5 Genetic algorithms

For the sake of simplicity of the exposition we will consider a maximization problem instead of a minimization problem in this section:

$$\max_x \; f(x) \quad \text{s.t. } x \in \mathscr{G}$$

where $\mathscr{G}$ is the feasible set. Furthermore, we assume that $f(x)$ is positive for any $x \in \mathscr{G}$.

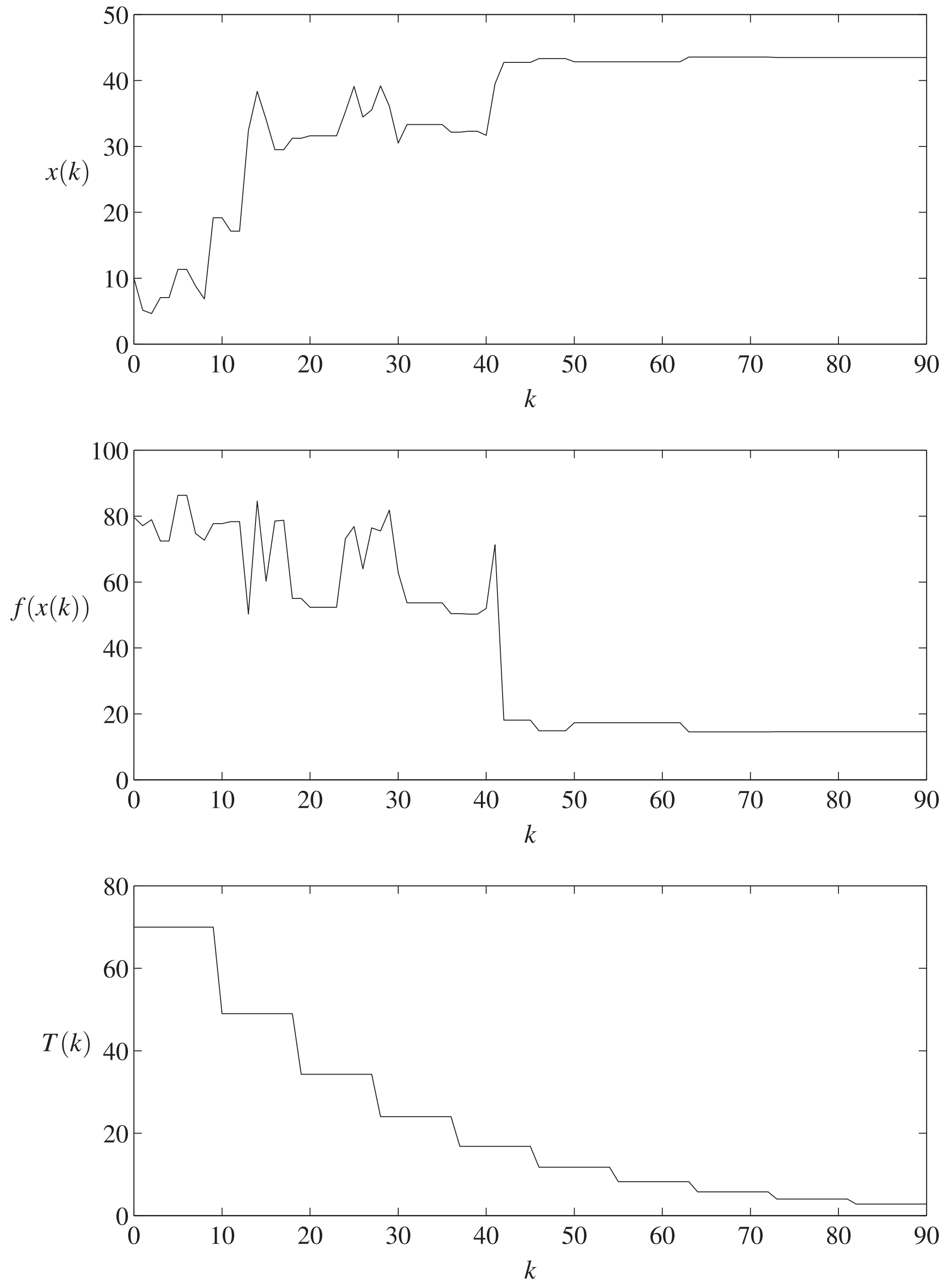


Figure 7.3: The current iteration point, its function value and the temperature for a typical run of the simulated annealing algorithm for the minimization problem of Example 7.1 with $x(0) = 10$, $T(0) = 70$, $m = 10$ and $\alpha = 0.7$.

Genetic algorithms [11, 18] try to find a global maximum by mimicking evolution in biology. The value of the objective function $f(x)$ then corresponds to the fitness of the individual solution x. We start with a group of solutions and then we use a biological process to maximize the fitness of the solutions: individual solutions are paired and create offspring in proportion to their fitness, there is cross-over between the parent solutions, and mutation can also take place.

Before we start the genetic algorithm, the solutions have to be encoded. The most widely used coding uses binary strings. Note however that other encodings (like, e.g., conventional decimal coding) are also possible. Also note that we are now in fact considering a feasible set $\mathscr{G}$ that is discretized (if the original feasible set is continuous, we can always define a grid to obtain a discretized set).

The basic genetic algorithm can be formulated as follows:

- **Initialization**:
  set up the initial population, i.e., create a set of binary strings $\{b_1, b_2, \ldots, b_N\}$ of equal length that are, e.g., uniformly distributed over $\mathscr{G}$, and evaluate their fitness.

- **Iteration**:
  create a new generation:

  - select the "parents":
    The probability that a string is selected is proportional to its relative fitness[2] $\dfrac{f(b_i)}{\sum_i f(b_i)}$.
  - create off-spring:
    There exist many techniques to create a new string from two given strings. One possibility is to use 1-point cross-over, where we first select a string position and then interchange the parts after the selected string position. Consider, e.g., two binary strings of length 5:

    $$\begin{aligned} b_1 &= 1\,0\,1\,0\,1 \\ b_2 &= 0\,0\,1\,1\,0 \ . \end{aligned}$$

    For string position 3, 1-point cross-over yields the following offspring:

    $$\begin{aligned} b_{1,\text{offspring}} &= 1\,0\,1\big|1\,0 \\ b_{2,\text{offspring}} &= 0\,0\,1\big|0\,1 \ . \end{aligned}$$

  - mutation:
    flip the bits of the offspring with probability p_{mutation}. The string 1 0 1 1 1 could then, e.g., be transformed into $1\,\underline{1}\,1\,1\,1$.

- **Stop** if the maximum number of generations is reached.

Genetic algorithms exhibit the following characteristics:

- They work with a coding of the parameter set: in order to be able to perform the genetic operations the points of the parameter set are encoded, usually by binary strings.

- They do not use derivatives; only the function values are used.

[2]For the sake of brevity of notation we assume here that f takes a string as argument and produces a real-valued number as result.

- They search from a population of points: instead of considering one point at the time as is done in random search, multi-start local optimization or simulation annealing, genetic algorithms consider sets of possible solutions in each iteration step.
- They can escape from local minima since a whole population of possible solutions is considered.
- They use probabilistic transition rules.

Let us now illustrate the basic genetic algorithm by an example.

Example 7.2 Consider the following (discrete) optimization problem:

$$\begin{aligned} &\max_x f(x) = \max_x \ 85 + (1-x)\sin\frac{x}{5} + x\cos\frac{x}{2} \\ &\text{s.t. } x \in \{0,1,\ldots,63\} \end{aligned}$$

The graph of f is represented in Figure 7.2.

First we code the feasible points as binary strings. Since there are only 64 elements in the feasible set $\mathscr{G} = \{0,1,\ldots,63\}$, we can encode each $x \in \mathscr{G}$ as a 6-bit string. The point $x = 24 = 16+8 = 2^4+2^3$ will, e.g., be encoded as 0 1 1 0 0 0.

Suppose that the initial population is given by the points of following table:

i	x_i	b_i	$f(x_i)$	$\frac{f(x_i)}{\sum_i f(x_i)}$
1	25	0 1 1 0 0 1	132.96	0.20
2	23	0 1 0 1 1 1	117.98	0.18
3	15	0 0 1 1 1 1	88.22	0.14
4	6	0 0 0 1 1 0	74.40	0.11
5	38	1 0 0 1 1 0	86.76	0.13
6	62	1 1 1 1 1 0	151.82	0.23

The average fitness for this population is 108.69, and the maximal fitness is 151.82.

Now we select parents p_i with a probability that is proportional to their relative fitness $\frac{f(x_i)}{\sum_i f(x_i)}$. Suppose that this results in $p_1 = b_1$, $p_2 = b_2$, $p_3 = b_2$, $p_4 = b_6$, $p_5 = b_6$ and $p_6 = b_5$. The offspring is generated using 1-point cross-over as indicated in the following table:

i	parent p_i	child c_i
1	$0\ 1\ 1\ 0\ 0 \mid 1$	$0\ 1\ 1\ 0\ 0 \mid 1$
2	$0\ 1\ 0\ 1\ 1 \mid 1$	$0\ 1\ 0\ 1\ 1 \mid 1$
3	$0\ 1\ 0 \mid 1\ 1\ 1$	$0\ 1\ 0 \mid 1\ 1\ 0$
4	$1\ 1\ 1 \mid 1\ 1\ 0$	$1\ 1\ 1 \mid 1\ 1\ 1$
5	$1\ 1 \mid 1\ 1\ 1\ 0$	$1\ 1 \mid 0\ 1\ 1\ 0$
6	$1\ 0 \mid 0\ 1\ 1\ 0$	$1\ 0 \mid 1\ 1\ 1\ 0$

Next there will be mutation. Suppose that the probability for mutation is equal to $p_{\text{mutation}} = 0.01$ per bit. This means that for the current population the expected number of bits that will be mutated is $0.01 \cdot 6 \cdot 6 = 0.36$ bits. Suppose that c_5 is mutated as follows:

$$c_5 = 110\underline{1}10 \quad \rightarrow \quad 110010 \ .$$

This leads to the new generation given in the following table:

i	x_i	b_i	$f(x_i)$	$\frac{f(x_i)}{\sum_i f(x_i)}$
1	25	0 1 1 0 0 1	132.96	0.19
2	23	0 1 0 1 1 1	117.98	0.17
3	22	0 1 0 1 1 0	105.08	0.14
4	63	1 1 1 1 1 1	145.69	0.20
5	50	1 1 0 0 1 0	161.22	0.23
6	46	1 0 1 1 1 0	50.46	0.07

The average fitness is now 118.90 and the maximal fitness in the population equals 161.22.

So we see that both the average and the maximal fitness have increased. Now we repeat the process and create new generations until the maximum number of generations (i.e., iterations) is reached; then we return the fittest individual as the final solution. $\square$

Chapter 8

Optimization Methods: Summary

In this chapter we discuss how the material presented in the previous chapters should be used when we are confronted with the following practical problem:

> Given an optimization problem and a (limited) set of available optimization algorithms (e.g., the MATLAB optimization toolbox (see Chapter 9)), select the most efficient algorithm from the given set of algorithms for the given optimization problem.

This problem can be solved in three steps:

Step 1: Simplification of the objective function and/or the constraints.

Step 2: Determination of the most efficient available algorithm.

Step 3: Determination of the stopping criterion.

In the next sections we discuss these steps in more detail.

8.1 Simplification of the objective function and/or the constraints

Sometimes an objective function or a constraint that at first sight may seem to be nonlinear and non-convex can be simplified (without changing the position of the optimum, of course). Unfortunately, there are no general, exhaustive rules for this simplification. Therefore, we will only illustrate this step by some examples.

Example 8.1 Consider the following minimization problem

$$\begin{aligned}
&\min_{x\in\mathbb{R}^3} f(x) = \min_{x\in\mathbb{R}^3} 2\exp(5x_1+6x_2-8x_3+3)\\
&\text{s.t. } 1 \leqslant 2x_1+3x_2+x_3 \leqslant 5\\
&\qquad\qquad\qquad x_1,x_2,x_3 \geqslant 0\,.
\end{aligned}$$

Since we are looking for a minimum, the factor 2 in the objective function does not influence the position of the minimum. Furthermore, since $\exp(\cdot)$ is an increasing function of its argument, minimizing the exponential function or its argument are equivalent. Finally, the constant term 3 also does not influence the position of the minimum. This implies that the nonlinear minimization problem given above can be simplified to the following linear programming problem:

$$\begin{aligned} &\min_{x\in\mathbb{R}^3} 5x_1+6x_2-8x_3 \\ &\text{s.t. } 1 \leqslant 2x_1+3x_2+x_3 \leqslant 5 \\ &\qquad\qquad x_1,x_2,x_3 \geqslant 0\,, \end{aligned}$$

which can be solved much more efficiently than the original problem. □

The following propositions (see also [10, 45]) show that optimization problems that involve 1-norms or ∞-norms of linear expressions and that have linear constraints, can also be recast as a linear programming problem.

Proposition 8.2 *Consider $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ and $w \in \mathbb{R}^n$. If $w_i > 0$ for all i, then the following problems are equivalent:*

(1) $\displaystyle\min_{x\in\mathbb{R}^n} \sum_{i=1}^{n} w_i|x_i|$ *subject to* $Ax \leqslant b$;

(2) $\displaystyle\min_{x,\alpha\in\mathbb{R}^n} \sum_{i=1}^{n} w_i\alpha_i$ *subject to* $Ax \leqslant b$, $\alpha \geqslant x$ *and* $\alpha \geqslant -x$.

Proof: We shall show that if problem (1) has an optimal solution $x^\star$, then problem (2) has an optimal solution $(x^\star,\alpha^\star)$ with $\alpha^\star = |x^\star|$ and vice versa.

Define $F_1(x) = \sum_{i=1}^{n} w_i|x_i|$ and $F_2(\alpha) = \sum_{i=1}^{n} w_i\alpha_i$. Note that the constraints $\alpha \geqslant x$ and $\alpha \geqslant -x$ are equivalent to the constraint $\alpha \geqslant |x|$. So if $\bar{x}$ is a feasible solution of problem (1) then $(\bar{x},\bar{\alpha})$ with $\bar{\alpha} = |\bar{x}|$ is a feasible solution of problem (2). On the other hand, if $(\hat{x},\hat{\alpha})$ is a feasible solution of problem (2), then $\hat{x}$ is a feasible solution of problem (1). Furthermore, since both F_1 and F_2 are bounded from below by 0 on the feasible set of the corresponding problem, problem (1) has an optimal solution if and only if problem (2) has an optimal solution.

Let $x^\star$ be an optimal solution of problem (1), and let $(\tilde{x},\tilde{\alpha})$ be an optimal solution of problem (2). Define $\alpha^\star = |x^\star|$. Note that $(x^\star,\alpha^\star)$ is a feasible solution of problem (2) and that $\tilde{x}$ is a feasible solution of problem (1). Furthermore, $F_1(x^\star) = F_2(\alpha^\star)$.

First we show that $(\tilde{x},\tilde{\alpha})$ is also an optimal solution of problem (1). Since $\tilde{\alpha} \geqslant \tilde{x}$ and $\tilde{\alpha} \geqslant -\tilde{x}$, we have $\tilde{\alpha} \geqslant |\tilde{x}|$. Now suppose that there exists an index j such that $\tilde{\alpha}_j > |\tilde{x}_j|$. In that case we could replace $\tilde{\alpha}_j$ by $|\tilde{x}_j|$ and still have a feasible solution of problem (2), but with a lower value of the objective function F_2 since all the w_i's are positive. This is impossible since $(\tilde{x},\tilde{\alpha})$ is an optimal solution of problem (2). Hence, we have $\tilde{\alpha} = |\tilde{x}|$. So $F_1(\tilde{x}) = F_2(\tilde{\alpha})$. Since $(x^\star,\alpha^\star)$ is a feasible solution of problem (2), we have $F_2(\tilde{\alpha}) \leqslant F_2(\alpha^\star) = F_1(x^\star)$. So $F_1(\tilde{x}) \leqslant F_1(x^\star)$. Since $x^\star$ is an optimal solution of problem (1), this implies that $\tilde{x}$ is also an optimal solution of problem (1).

Now we show that $(x^\star,\alpha^\star)$ is also an optimal solution of problem (2). In the first part of the proof we have already shown that $F_1(\tilde{x}) = F_2(\tilde{\alpha})$ and $F_1(x^\star) = F_2(\alpha^\star)$. Since $\tilde{x}$ is a feasible solution of problem (1), we have $F_1(x^\star) \leqslant F_1(\tilde{x})$. Hence, $F_2(\alpha^\star) = F_1(x^\star) \leqslant F_1(\tilde{x}) = F_2(\tilde{\alpha})$. Since $(\tilde{x},\tilde{\alpha})$ is an optimal solution of problem (2), this implies that $(x^\star,\alpha^\star)$ is also an optimal solution of problem (2). □

Proposition 8.3 *Consider $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ and $w \in \mathbb{R}^n$. If $w_i \geqslant 0$ for all i, then the following problems are equivalent:*

(1) $\min_{x \in \mathbb{R}^n} \max_i w_i |x_i|$ *subject to* $Ax \leqslant b$*;*

(2) $\min_{x \in \mathbb{R}^n, t \in \mathbb{R}} t$ *subject to* $Ax \leqslant b$*,* $t \geqslant w_i x_i$ *and* $t \geqslant -w_i x_i$ *for all i.*

Proof: Note that $t \geqslant w_i x_i$ and $t \geqslant -w_i x_i$ for all i implies that $t \geqslant w_i |x_i|$ for all i and thus also $t \geqslant \max_i w_i |x_i|$. Now we can use a reasoning that is similar to the one of the proof of Proposition 8.2 to prove that problem (1) and problem (2) are equivalent. □

Remark 8.4 Note that if $w_i = 1$ for all i, then the objective functions of problem (1) in Propositions 8.2 and 8.3 correspond to respectively the 1-norm and the ∞-norm of the parameter vector x. ◇

8.2 Determination of the most efficient available algorithm

We assume that the problem has been simplified as far as possible and that the constraints that can be eliminated have been eliminated (cf. Sections 5.1 and 5.2). The most efficient algorithm can now be determined by going through the following steps:

- If the objective function and the constraints are linear, then the most efficient method is the simplex method, unless it is a large-sized problem since then an interior-point method will usually be more efficient (cf. Remarks 2.4 and 6.5).

- If the objective function is quadratic and the constraints are linear, then the most efficient method is the modified simplex method, unless it is a large-sized problem since then an interior-point method will usually be more efficient (cf. Remark 6.5).

- For convex problems we can use the cutting-plane algorithm, the ellipsoid algorithm, or an interior-point method.

- If the problem is known or suspected to have many local minima, we can use multi-start local optimization, simulated annealing, or a genetic algorithm.

- For nonlinear non-convex problems it is usually better to use the gradient and Hessian information if that can be computed efficiently (i.e., if analytic expressions are available). If no analytic expressions are available, then the gradient can be computed numerically using finite differences. Note that this should not be done if the evaluation of the objective function is computationally intensive.
  If we have an unconstrained problem then we use the following algorithms (in decreasing order of preference):

  - The Levenberg-Marquardt or the Newton algorithm (in its original version, or in its line search variant), provided that analytic expressions for the gradient and the Hessian are available or that the evaluation of the objective function is not computationally intensive at all so that the Hessian and the gradient can be computed numerically without too much computational effort.
  - A quasi-Newton algorithm (in its original version, or in its line search variant), provided that an analytic expression for the gradient is available or that the gradient can be computed numerically without too much computational effort.

  - A steepest descent algorithm, provided that an analytic expression for the gradient is available or that the gradient can be computed numerically without too much computational effort.
  - If no analytic expression for the gradient is available or if the numerical computation of the gradient would require too much computational effort, then we could use Powell's perpendicular method or — when that is not available and when the number of variables is small, say 2 or 3 — the Nelder-Mead method.

For constrained problems with linear constraints we can use the gradient projection algorithm or the SQP algorithm. For problems with nonlinear constraints we can either use the SQP algorithm or we can transform the problem into an unconstrained optimization problem using a penalty or barrier function approach.

8.3 Determination of the stopping criterion

In many cases the selection of the algorithm will immediately determine which stopping criterion should be used. Recall that stopping criteria have already been discussed extensively in Section 1.6. Some important remarks are:

- The simplex algorithm and the modified simplex algorithm can find the optimal solution in a finite number of steps. So in theory no stopping criterion is necessary.
- For constrained optimization problems *necessary* conditions for an extremum are given by the Lagrange conditions or the Kuhn-Tucker conditions.
- For the convex optimization methods (cutting-plane algorithm, ellipsoid algorithm, and interior-point method) the stopping criterion could use the distance of the current objective function to the *real* minimum of the objective function.
- In practice, a stopping criterion will consist of several components: it will consist of a mix of (absolute and/or relative) constraints on the objective function value, the current point and the number of iterations.

Chapter 9

The MATLAB Optimization Toolbox

This chapter provides a short introduction to the *MATLAB Optimization Toolbox*[1]. This toolbox contains functions that can be used to solve several optimization problems. The functions discussed in this chapter are those solving the linear programming problem, the quadratic programming problem, and the unconstrained as well as the constrained nonlinear optimization problem. The use of each function is illustrated with an example. For a description of the underlying algorithms we refer to the preceding chapters. The MATLAB toolbox offers algorithms to solve both large-scale and medium-scale problems. We will focus on the medium-scale algorithms. Note that not all functions contained in the MATLAB optimization toolbox are discussed here. More information can be found in either the *Optimization Toolbox User's Manual* [42] or the on-line help in MATLAB, which is accessible via the MATLAB command `doc`. The optimization toolbox also contains several illustrative demos. Typing `optdemo` at the MATLAB prompt yields a menu in which the demos can be selected.

9.1 Linear programming

The linear programming problem

$$\min_x c^T x \qquad \text{subject to } Ax \leqslant b$$

can be solved using the function `linprog`. For medium-scale problems this function uses the simplex method (see Chapter 2) or an active-set method (which is a variation of the simplex method), and for large-scale problems a primal-dual interior-point method is used (cf. Section 6.4.3). The syntax of `linprog` is:

```
    x=linprog(c,A,b,Aeq,beq,lb,ub,x0,options)
```

where

[1]This chapter describes Version 6.4 of the MATLAB Optimization Toolbox present in MATLAB release R2013b. The most recent version of the optimization toolbox is described at `www.mathworks.com/products/optimization/`. In particular, options and implementations may vary from one release of the toolbox to another.

`x`	= Vector that minimizes the objective function.
`c`	= Constant coefficients of the objective function.
`A, b`	= Coefficients of the inequality constraints ($\texttt{A x} \leqslant \texttt{b}$).
`Aeq, beq`	= Coefficients of the equality constraints ($\texttt{Aeq x} = \texttt{beq}$). These arguments are optional. By default no equality constraints are imposed.
`lb, ub`	= Lower and upper bounds for `x`, i.e., $\texttt{lb} \leqslant \texttt{x} \leqslant \texttt{ub}$. These arguments are optional. By default no bounds are imposed.
`x0`	= Initial starting point. This argument is optional. Its default value is a vector of zeros. The provided starting point is ignored if either a large-scale algorithm is selected or if the simplex algorithm is selected (see below).
`options`	= Options structure. This argument is optional.

The most important options are:

`Algorithm`	: Selects the algorithm to be used. Possible values are `interior-point` (this is the default), `active-set`, and `simplex`.
`Diagnostics`	: Prints diagnostic information about the problem to be solved if set to `on`. The default value is `off`.
`Display`	: Controls the display of (intermediate) values. Possible values are: `off` (displays no output), `iter` (displays output at each iteration[2]), `final` (displays just the final output). The default is `final`.
`MaxIter`	: Maximum number of iterations allowed (the default value depends on the algorithm selected, and is, e.g., 10 $\times$ number of variables for the simplex algorithm).

The options can be set with the command `optimoptions` where the first argument is the optimization function used, e.g.,

```
options=optimoptions('linprog','Algorithm','simplex','on',...
                     'Display','off')
```

The default values of the options for the linear programming algorithms can be retrieved by the MATLAB command

```
default_linprog_options=optimoptions('linprog')
```

It is also possible to use extra output arguments to obtain more information about the optimization process and the final result. The command

```
[x,fval,exitflag,output,lambda]=linprog(c,A,b,Aeq,beq,lb,ub,x0,options)
```

also returns the final value of the objective function `fval`, and a flag variable `exitflag` describing the exit condition (> 0 if the algorithm converged to a solution `x`, 0 if the maximum number of function evaluations or iterations was exceeded, and < 0 if the algorithm did not converge to a solution). The structure `output` contains among others the number of iterations (`output.iterations`), and the structure `lambda` contains the Lagrange multipliers at the solution `x`.

Example 9.1 The use of the function `linprog` will be illustrated using the brewery scheduling problem presented in Example 2.1. The problem can be stated as follows:

$$\min_{x_1,x_2} -20x_1 - 30x_2,$$

[2]At this time, the `iter` setting only works with the interior-point algorithm and the simplex algorithm.

subject to the constraints

$$\begin{aligned} x_1 + x_2 &\leqslant 100 \\ 0.1x_1 + 0.2x_2 &\leqslant 14 \\ x_1, x_2 &\geqslant 0 \end{aligned}$$

It is easy to verify that this optimization problem can be solved by the following sequence of MATLAB commands:

```
c=[-20 -30];
A=[  1   1
   0.1 0.2];
b=[100 14]';
lb=zeros(2,1);
ub=[];           % No upper bound
options=optimoptions('linprog','Algorithm','simplex');
                 % Select the simplex algorithm.
x=linprog(c,A,b,[],[],lb,ub,[],options)
profit=-c*x
```

□

9.2 Quadratic programming

The quadratic programming problem

$$\min_x \frac{1}{2}x^T Hx + c^T x \qquad \text{subject to } Ax \leqslant b$$

can be solved using the function `quadprog`. For medium-scale problems this function uses a variation of the modified simplex method (see Chapter 3). For large-scale problems the function uses the subspace trust-region method based on the interior-reflective Newton method described in [9]. The syntax of `quadprog` is:

```
x=quadprog(H,c,A,b,Aeq,beq,lb,ub,x0,options)
```

where

`x`	=	Vector that minimizes the objective function.
`H, c`	=	Constant coefficients of the objective function.
`A, b`	=	Coefficients of the inequality constraints (`A x` $\leqslant$ `b`)
`Aeq, beq`	=	Coefficients of the equality constraints (`Aeq x = beq`). These arguments are optional. By default no equality constraints are imposed.
`lb, ub`	=	Lower and upper bounds for `x`, i.e., `lb` $\leqslant$ `x` $\leqslant$ `ub`. These arguments are optional. By default no bounds are imposed.
`x0`	=	Initial starting point. This argument is optional. Its default value is a vector of zeros. The provided starting point is only used with the active-set algorithm.
`options`	=	Options structure. This argument is also optional.

The most important options are:

`Algorithm`	: Selects the algorithm to be used. Possible values are `trust-region-reflective` (default; note however that this typically is a large-scale algorithm), `interior-point-convex`, and `active-set`.
`Diagnostics`	: Prints diagnostic information about the problem to be solved if set to `on`. The default value is `off`.
`Display`	: Controls the display of (intermediate) values. Possible values are: `off` (displays no output) and `final` (displays just the final output). The default is `final`.
`MaxIter`	: Maximum number of iterations allowed (default value: 200 for the interior-point and the active-set algorithm).

Just as for `linprog` you could also use the extended syntax

```
[x,fval,exitflag,output,lambda]= quadprog(H,c,A,b,Aeq,beq,lb,ub,x0,options)
```

to obtain more information about the optimization process and the final result (see Section 9.1 for more explanation about the meaning of the output arguments).

Example 9.2 The use of the function `quadprog` will be illustrated using the system identification example presented in Section 3.2. The system to be identified is given by

$$y(n+1)+ay(n)=bu(n)+e(n)$$

where $u(n)$ is the input signal, $y(n)$ is the output signal, and $e(n)$ is an unknown noise signal. Given the input data sequence $u(1),u(2),\ldots,u(4)$ and the output data sequence $y(1),y(2),\ldots,y(5)$ of this system, we want to find estimates $\hat{a}$ and $\hat{b}$ of the system parameters a and b. As shown in Example 3.6, we have the following optimization problem

$$\min_x (Y-\Phi x)^T(Y-\Phi x)=\min_x x^T\Phi^T\Phi x-2\Phi^T Yx+Y^TY$$

where

$$Y=\begin{bmatrix} y(2)\\ y(3)\\ y(4)\\ y(5)\end{bmatrix},\qquad \Phi=\begin{bmatrix} -y(1) & u(1)\\ -y(2) & u(2)\\ -y(3) & u(3)\\ -y(4) & u(4)\end{bmatrix},\qquad x=\begin{bmatrix}\hat{a}\\ \hat{b}\end{bmatrix}$$

under the constraint $-0.99\leqslant\hat{a}\leqslant 0.99$. With the data sequences given in Example 3.6 this optimization problem can be solved as follows:

```
y=[ 0  0.74  -0.8  1.97  -1.09 ]';
u=[ 0.33  -0.12  0.8  -0.07]';
Y=y(2:5);
Phi=[-y(1:4) u];
H=2*Phi'*Phi;
c=-2*Phi'*Y;
lb=[-0.99 -Inf]';
ub=[0.99 Inf];
options=optimoptions('quadprog','Algorithm','active-set');
     % Select the active-set algorithm.
x=quadprog(H,c,[],[],[],[],lb,ub,[],options)
energy=0.5*x'*H*x+c'*x+Y'*Y
```

□

Example 9.3 Model Predictive Control (MPC). In this example we show how `quadprog` can be used to design a simple model predictive controller. Consider a controllable, SISO, linear, time-invariant, discrete-time state space model:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned}$$

where

$$A = \begin{bmatrix} 2.4936 & -1.2130 & 0.4176 \\ 2.0000 & 0 & 0 \\ 0 & 1.0000 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0.2500 \\ 0 \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0.0684 & 0.1288 & 0.0313 \end{bmatrix}$$

The control objective is to steer the output of the plant to zero, and to keep it there. In addition, we want to keep the control signal $u(k)$ small. It is assumed that we can measure the state of the system. We define the performance index over a prediction horizon N_p:

$$J(k) = \sum_{i=1}^{N_\text{p}} y(k+i|k)^2 + \lambda\, u(k+i-1|k)^2$$

where λ determines the trade-off between controller action and output regulation, and $y(k+i|k)$ is the output at time $k+i$ predicted from the information that is available at time k. At time instant k we compute an input sequence $u(k|k)$, $u(k+1|k)$,…,$u(k+N_\text{p}-1|k)$ such that $J(k)$ is minimized and $|u(k+i-1|k)| \leqslant 0.25$ for $i = 1,\ldots,N_\text{p}$. Therefore, at sample step k we need to compute predictions of the output y at the sample steps $k+i$, $i = 1,2,\ldots,N_\text{p}$. After computing these predictions, the control signal $u(k|k)$ is implemented at sample step k. At the next sample step, $k+1$, we do not implement the control signal $u(k+1|k)$ from the previous calculations, but we shift the horizon N_p one step forward in time and we adapt the estimate of the current state (and possibly also the parameters of the model) based on measurements of the system, and we compute the necessary predictions of the output. These predictions are used to compute the optimal control signal at sample step $k+1$. In this way, the predictions at a certain time instant are based on all the information that is available at that time instant, therefore the control strategy is less sensitive to a possible mismatch between the model and the real plant. This control strategy is called the *receding horizon* approach.

To minimize $J(k)$ we need predictions of $x(k)$ and $y(k)$. The prediction of $x(k)$ up to the horizon N_p is given by:

$$\begin{bmatrix} x(k+1|k) \\ \vdots \\ x(k+N_\text{p}|k) \end{bmatrix} = \begin{bmatrix} A \\ \vdots \\ A^{N_\text{p}} \end{bmatrix} x(k) + \begin{bmatrix} B & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ A^{N_\text{p}-1}B & \cdots & \cdots & B \end{bmatrix} \begin{bmatrix} u(k|k) \\ \vdots \\ u(k+N_\text{p}-1|k) \end{bmatrix}$$

or equivalently:

$$\tilde{x} = \tilde{A}x(k) + \tilde{B}\tilde{u}$$

The prediction of $y(k)$ up to the horizon N_p is given by:

$$\begin{bmatrix} y(k+1|k) \\ \vdots \\ \vdots \\ y(k+N_\text{p}|k) \end{bmatrix} = \begin{bmatrix} C & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & C \end{bmatrix} \begin{bmatrix} x(k+1|k) \\ \vdots \\ \vdots \\ x(k+N_\text{p}|k) \end{bmatrix}$$

or equivalently:

$$\tilde{y} = \tilde{C}\tilde{x}$$

The performance index over the horizon N_p can be written as:

$$\begin{aligned} J(k) &= \tilde{y}^T\tilde{y} + \lambda\,\tilde{u}^T\tilde{u} \\ &= \underbrace{x^T(k)\tilde{A}^T\tilde{C}^T\tilde{C}\tilde{A}x(k)} + 2\tilde{u}^T\tilde{B}^T\tilde{C}^T\tilde{C}\tilde{A}x(k) + \tilde{u}^T(\tilde{B}^T\tilde{C}^T\tilde{C}\tilde{B} + \lambda I_{N_\mathrm{p}})\tilde{u} \end{aligned}$$

Note that the under-braced term does not depend on $u(k)$; therefore, we can neglect it in the optimization procedure.

The MATLAB code below implements the MPC controller and simulates the closed-loop system for $k = 0, 1, \ldots, 50$. We take $N_\mathrm{p} = 10$, $\lambda = 1$, and $x(0) = [1\ 1\ 1]^T$. Figure 9.1 shows the resulting control signal $u(k)$. Figure 9.2 shows the corresponding output signal $y(k)$.

```
% Model Predictive Control Example

% Define the system matrices.
A=[2.4936 -1.2130 0.4176
   2       0      0
   0       1      0];
B=[0.25 0 0]';
C=[0.0684 0.1288 0.0313];
D=0;

Np=10;        % Prediction and control horizon
lambda=1;     % Trade-off parameter
x0=[1 1 1]';  % Initial state

% Construct the prediction matrices.
At=[];
for i=1:Np,
  At=[At;A^i];
end;

temp=[];
Bt=[];
for i=1:Np,
  Bt=[Bt        zeros(size(Bt,1),size(B,2))
      A^(i-1)*B temp];
  temp=[A^(i-1)*B temp];
end;

Ct=[];
for i=1:Np,
  Ct=[Ct zeros(size(Ct,1),size(C,2))
      zeros(size(C,1),size(Ct,2)) C];
end;

% Define the lower and upper bounds, and the initial value.
lb=-0.25*ones(Np,1);
ub=0.25*ones(Np,1);
u0=zeros(Np,1);
```

```
% Simulation of the closed-loop system
u=[];
x=x0;     % Initial state
y=C*x0;   % Initial output
xk=x0;    % Start of the horizon
H=2*((Ct*Bt)'*(Ct*Bt)+lambda*eye(Np,Np)); % Note that H is a constant
                                          % matrix that does not depend
                                          % on k.


options=optimoptions('quadprog','Algorithm','active-set');
                                    % Select the active-set algorithm

for k=1:50,
  ut=quadprog(H,2*xk'*At'*Ct'*Ct*Bt,[],[],[],[],lb,ub,u0,options);
  u(k)=ut(1);                    % Implement only the 1st control
                                 % sample.
  x(:,k+1)=A*x(:,k)+B*u(k);      % Compute the state.
  y(:,k+1)=C*x(:,k+1);           % Compute the output.
  xk=x(:,k+1);                   % Shift the horizon.
  u0=[ut(2:Np);ut(Np)];          % Use the shifted solution as the
                                 % initial solution for the next
                                 % iteration.
end;
```

□

9.3 Unconstrained nonlinear optimization

Consider the unconstrained nonlinear optimization problem

$$\min_x f(x)$$

where f is a nonlinear function. The MATLAB functions `fminunc` and `fminsearch` can be used to solve this optimization problem. For medium-scale problems the function `fminunc` uses an algorithm that is based on direction determination and line search (see Section 4.2). For large-scale problems a subspace trust region method is used (see [7, 8]). The function `fminsearch` implements the Nelder-Mead, nonlinear simplex method (see Section 4.3).

The syntax of `fminunc` is:

```
x=fminunc(@fun,x0,options)
```

where

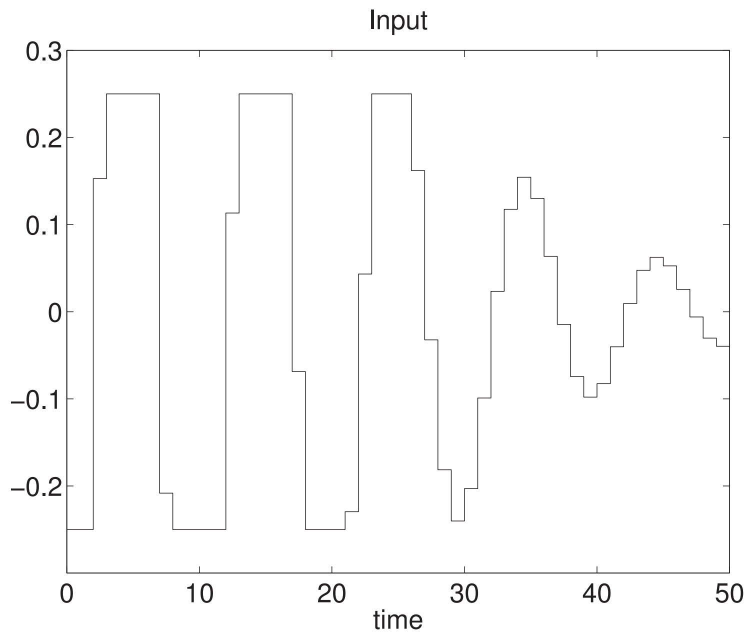


Figure 9.1: MPC control signal of Example 9.3.

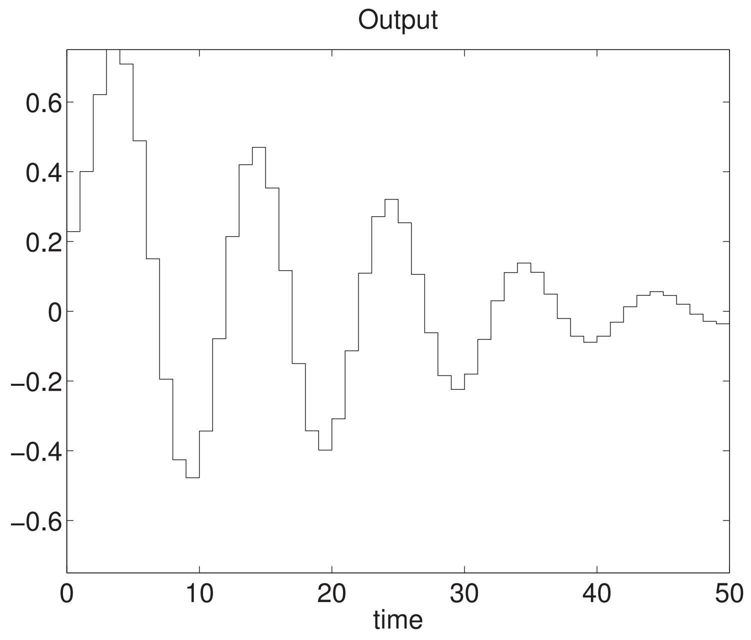


Figure 9.2: Output of the MPC controlled plant of Example 9.3.

`fun`	= The name of an m-file describing the objective function. The function `fun` should return a scalar function value `f` in the given point `x` as its first output argument. In this case the calling syntax of `fun` is `f=fun(x)`. If the gradient of `fun` can also be computed and the option `GradObj` is set to `on`, then the function `fun` must return the gradient vector `g` as second output argument. The calling syntax of `fun` is then `[f,g]=fun(x)`. The gradient must be provided to use the large-scale method. It is optional for the medium-scale method (if it is not provided by the user a finite-difference approximation will be used).
`x0`	= Initial starting point.
`options`	= Options. This argument is optional.

The most important options for `fminunc` are:

`Algorithm`	: Selects the algorithm to use. Possible values are `trust-region` (default; note that this is a large-scale algorithm) and `quasi-newton`.
`DerivativeCheck`	: If set to `on`, then the user-supplied derivatives (gradients) will be compared to finite differencing derivatives. The default value is `off`.
`Diagnostics`	: Prints diagnostic information about the problem to be solved if set to `on`. The default value is `off`.
`Display`	: Controls the display of (intermediate) values. Possible values are: `off` (no output), `iter` (displays output at each iteration), `notify` (displays output only if `fminunc` does not converge), and `final` (displays just the final output). The default value is `final`.
`GradObj`	: Indicates whether the gradient for the objective function is defined by user. The default value is `off`, which causes `fminunc` to estimate gradients using a finite-difference method.
`MaxFunEvals`	: Maximum number of function evaluations allowed (default value: 100 $\times$ number of variables).
`MaxIter`	: Maximum number of iterations allowed (default value: 400).
`TolFun`	: Termination tolerance on the function value (default value: 10^{-6}).
`TolX`	: Termination tolerance on `x` (default value: 10^{-6}).

In addition, the following option can be used for the algorithm based on direction determination and line search (i.e., when the `Algorithm` option is set to `quasi-newton`:

`HessUpdate`	: `bfgs` (default) selects the Broyden-Fletcher-Goldfarb-Shanno formula for updating the approximation of the Hessian matrix. The Davidon-Fletcher-Powell formula is selected by setting this option to `dfp`. A steepest-descent method is selected by setting it to `steepdesc`.

It is also possible to use extra output arguments to obtain more information about the optimization process. The command

```
[x,fval,exitflag,output]=fminunc(@fun,x0)
```

also returns the final value of the objective function `fval`, and a flag variable `exitflag` describing the exit condition (> 0 if the algorithm converged to a solution `x`, 0 if the maximum number of function evaluations or iterations was exceeded, and < 0 if the algorithm did not converge to a solution). The structure `output` contains among others the number of function evaluations (`output.funcCount`), and the number of iterations (`output.iterations`).

The syntax of `fminsearch` is:

```
x=fminsearch(@fun,x0,options)
```

where

`fun`	=	The name of an m-file describing the objective function. The function `fun` should return a scalar function value: `f=fun(x)`
`x0`	=	Initial starting point.
`options`	=	Options. This argument is optional.

The structure `options` allows the user to control the properties of the optimization process. For `fminsearch` we have the following option fields:

`Display`	:	Level of display: `off` displays no output; `iter` displays output at each iteration; `final` (default) displays just the final output; `notify` displays output only if the function does not converge.
`MaxFunEvals`	:	Maximum number of function evaluations allowed (default value: 200 $\times$ number of variables).
`MaxIter`	:	Maximum number of iterations allowed (default value: 200 $\times$ number of variables).
`TolFun`	:	Termination tolerance on the function value (default value: 10^{-4}).
`TolX`	:	Termination tolerance on `x` (default value: 10^{-4}).

Example 9.4 We will now use the MATLAB functions described in this section to find a minimum of Rosenbrock's function, which is given by:

$$f(x_1,x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

First, we write an m-file called `fun_rosenbrock.m` for this function

```
function f=fun_rosenbrock(x)
f=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
```

The solution can be calculated using the Nelder-Mead method, as follows:

```
x0=[1.5 1.5]';
x=fminsearch(@fun_rosenbrock,x0)
```

It is also possible to use a method with direction determination and line search, e.g., the Davidon-Fletcher-Powell direction combined with mixed quadratic and cubic polynomial interpolation.

```
x0=[1.5 1.5]';
% Select the quasi-Newton algorithm with Davidon-Fletcher-Powell
% direction.
options=optimoptions('fminunc','Algorithm','quasi-newton',...
   'HessUpdate','dfp');
x=fminunc(@fun_rosenbrock,x0,options)
```

Since no gradient is supplied in this example, it will be approximated by finite differences. However, the gradient of Rosenbrock's function can easily be calculated. It is given by

$$\nabla f(x_1,x_2) = \begin{bmatrix} 400x_1^3 - 400x_1x_2 + 2x_1 - 2 \\ 200x_2 - 200x_1^2 \end{bmatrix}$$

We now adapt the m-file `fun_rosenbrock` to include the computation of the gradient:

```
function [f,g]=fun_rosenbrock(x)
f=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
%
% If the function is called with 2 output arguments then
% also compute the gradient.
%
if ( nargout > 1 )
   g=[ 400*x(1)^3-400*x(1)*x(2)+2*x(1)-2
        200*x(2)-200*x(1)^2];
end;
```

Now the minimum can be calculated without approximation of the gradient as follows

```
x0=[1.5 1.5]';
% Select the quasi-Newton algorithm with Davidon-Fletcher-Powell
% direction.
options=optimoptions('fminunc','Algorithm','quasi-newton',...
   'HessUpdate','dfp');
% Also indicate that the gradient will be computed if necessary.
options=optimoptions(options,'GradObj','on');
x=fminunc(@fun_rosenbrock,x0,options)
```

□

Nonlinear least squares problems

Consider the unconstrained nonlinear least squares problem

$$\min_x e^T(x)e(x)$$

where $e(x)$ is a nonlinear vector function. The MATLAB function `lsqnonlin` can be used to solve this optimization problem. For large-scale problems this function uses a subspace trust region method based on the interior-reflective Newton method described in [7, 8]. For medium-scale problems the Levenberg-Marquardt method is used.

The syntax of `lsqnonlin` is:

```
x=lsqnonlin(@fun,x0,lb,ub,options)
```

where

fun	= The name of an m-file describing the objective function. The function fun should return the *vector e*: e=fun(x). If the Jacobian ∇e can also be computed and the option Jacobian is set to on then the function fun must return, in a second output argument, the ***transposed*** Jacobian in J: [e,J]=fun(x). If the Jacobian is not provided by the user, then lsqnonlin approximates the Jacobian using finite differences.
x0	= Initial starting point.
lb, ub	= Lower and upper bounds for x. Optional. Note that the Levenberg-Marquardt algorithm does *not* handle bound constraints
options	= Options structure. Optional.

The most important options for lsqnonlin are:

Algorithm	: Selects the algorithm to be used. Possible values are trust-region-reflective (default; note that this is a large-scale algorithm) and levenberg-marquardt.
DerivativeCheck	: If set to on, then the user-supplied derivatives (Jacobian) will be compared to finite differencing derivatives. The default value is off.
Diagnostics	: Prints diagnostic information about the problem to be solved if set to on. The default value is off.
Display	: Controls the display of (intermediate) values. Possible values are off and final.
Jacobian	: Indicates whether the Jacobian is defined by user.
MaxFunEvals	: Maximum number of function evaluations allowed (default value: 100 $\times$ number of variables).
MaxIter	: Maximum number of iterations allowed (default value: 400).
TolFun	: Termination tolerance on the function value (default value: 10^{-6}).
TolX	: Termination tolerance on x (default value: 10^{-6}).

Example 9.5 To illustrate the use of lsqnonlin we will again use Rosenbrock's function (see Example 9.4). Note that $e(x)$ becomes

$$e(x_1,x_2) = \begin{bmatrix} 10(x_2 - x_1^2) & (1 - x_1) \end{bmatrix}^T$$

First, we write an m-file fun_rosenbrock_e.m for this function and its transposed Jacobian.

```
function [e,J]=fun_rosenbrock_e(x)
e=[10*(x(2)-x(1)^2) (1-x(1))]';
%
% If the function is called with 2 output arguments then
% also compute the transposed Jacobian.
%
if ( nargout > 1 )
   J=[-20*x(1)  10
      -1        0];
end;
```

Note that the ith column of J is the gradient of ith component of e (i.e., J is the transpose of ∇e). Finally, we solve the minimization problem using the Levenberg-Marquardt approach.

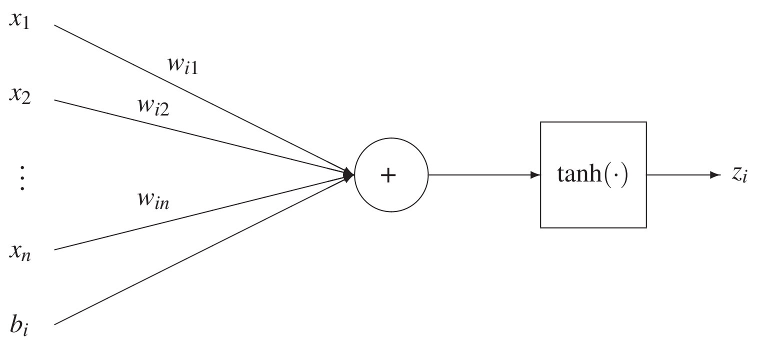


Figure 9.3: A neuron, with n inputs x_j, one output z_i, weights w_{ij} and bias b_i.

```
x0=[1.5 1.5]';
% Select Levenberg-Marquardt algorithm and indicate that the
% Jacobian is provided by the user.
options=optimoptions('lsqnonlin',...
   'Algorithm','levenberg-marquardt',...
   'Jacobian','on');
x=lsqnonlin(@fun_rosenbrock_e,x0,[],[],options)
```

□

Example 9.6 Identification using a neural network. In this example we use a feed-forward neural network to model a simple nonlinear dynamical system. We use the function `lsqnonlin` to train the neural network, i.e., to find its parameter values. A neural network consists of a number of *layers* containing *neurons*. Neurons are the basic processing units of the network. A neuron calculates a weighted sum of its inputs and adds a bias term to it (see Figure 9.3). The resulting output is mapped onto a nonlinear function, which is called the *activation function*. A commonly used activation function is the hyperbolic tangent:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Neurons can be grouped into interconnected layers. A feed-forward neural network consists of several uni-directional layers. All outputs of one layer are inputs for the next layer. This type of network does not contain interconnections between neurons in the same layer. The first layer of the network is called the *input layer*. It directs the inputs of the network to the next layer. The last layer of the network is called the *output layer*. The other layers are called *hidden layers*. Only the hidden layers contain nonlinear activation functions [6, 40].

In this example we will use a feed-forward neural network with one hidden layer and one output. The equations describing this network are given by:

$$\begin{aligned} v_j(k) &= \sum_{i=1}^{N^{\mathrm{i}}} w_{ij}^{\mathrm{h}} \phi_i(k) + b_j^{\mathrm{h}} \\ \hat{y}(k) &= \sum_{j=1}^{N^{\mathrm{h}}} w_j^{\mathrm{o}} \tanh\Big(v_j(k)\Big) + b^{\mathrm{o}} \end{aligned}$$

where $\phi_i(k)$ are the inputs to the neural network, $\hat{y}(k)$ is the output, $v_j(k)$ are auxiliary signals, N^{i} is the number of inputs, N^{h} is the number of hidden neurons, w_{ij}^{h} are the weights of the hidden layer,

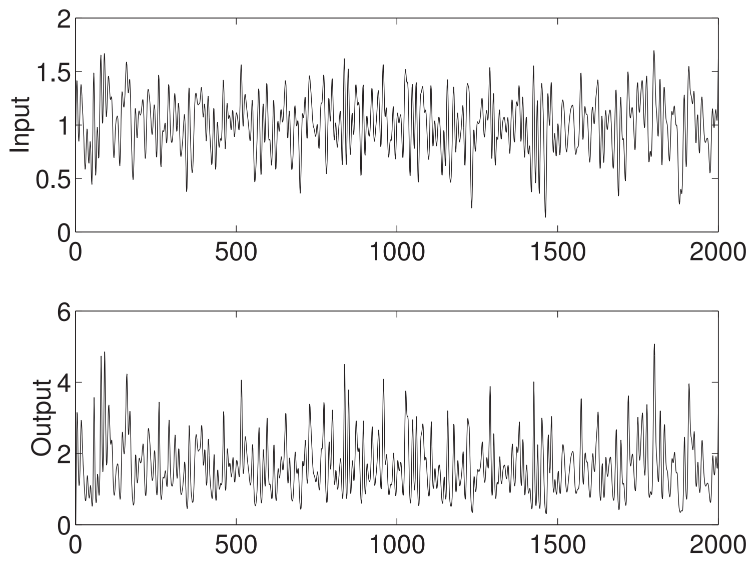


Figure 9.4: Input and output data used to train the neural network of Example 9.6.

b_j^{h} are the biases of the hidden layer, w_j^{o} are the weights of the output layer, and b^{o} is the bias of the output layer. If we take N samples of the input and output signals, we can also write the neural network equations as:

$$\begin{aligned} V &= \Phi W^{\mathrm{h}} + U_N B^{\mathrm{h}} \\ \hat{Y} &= \tanh(V) W^{\mathrm{o}} + U_N b^{\mathrm{o}} \end{aligned}$$

where U_N is an N-dimensional column vector with all its entries equal to one, $V \in \mathbb{R}^{N \times N^{\mathrm{h}}}$, $\Phi \in \mathbb{R}^{N \times N^{\mathrm{i}}}$, $\hat{Y} \in \mathbb{R}^{N \times 1}$, $W^{\mathrm{h}} \in \mathbb{R}^{N^{\mathrm{i}} \times N^{\mathrm{h}}}$, $B^{\mathrm{h}} \in \mathbb{R}^{1 \times N^{\mathrm{h}}}$, and $W^{\mathrm{o}} \in \mathbb{R}^{N^{\mathrm{h}} \times 1}$.

We use a neural network to model the following nonlinear system [28, 33]:

$$y(k+1) = \frac{y(k)}{1 + y^2(k)} + u^3(k)$$

We assume that we do not know this system, and that we only have 2000 samples of the input and output signal. These signals are shown in Figure 9.4. The neural network that we are going to use has four hidden neurons, and takes $u(k-1)$ and $y(k-1)$ as inputs. Define the error between the real output and the output of the neural network as

$$e(k) = y(k) - \hat{y}(k)$$

Now we have to solve the following minimization problem

$$\min_{W^{\mathrm{h}}, W^{\mathrm{o}}, B^{\mathrm{h}}, b^{\mathrm{o}}} \sum_{k=1}^{N} |e(k)|^2 \ .$$

The following m-file computes the optimum using the Levenberg-Marquardt method. We assume that the 2000 samples of the input and output data are stored in the variables `u` and `y` respectively.

```
% Optimization example -- Neural Network

N=size(u,1);                 % Number of data points
Nh=4;                        % Number of hidden neurons
Ni=2;                        % Number of input neurons
Phi=[y(2:N,1), u(2:N,1)]; % Regressors
Y=y(1:N-1,1);                % Output

% Initialize weights and biases.
x0=nninit(Phi,Y,Nh);  % Initial weights and biases

% Set the options.

% Select the Levenberg-Marquardt and indicate that the Jacobian
% is provided by the user.
options=optimoptions('lsqnonlin','Algorithm','levenberg-marquardt',...
                 'Jacobian','on');

% Also set termination tolerance on parameters and objective
% function.
options=optimoptions(options,'TolX',0.05,'TolFun',0.5);

% Train the neural network.
x=lsqnonlin(@(x) nnfun(x,Phi,Y,Nh),x0,[],[],options);
```

This m-file uses two other m-file functions `nninit.m` and `nnfun.m`, which are explained next. Also note that we have used the extended syntax

```
x=lsqnonlin(@(x) fun(x,P1,P2,...,Pn),x0,lb,ub,options)
```

which allows to pass the parameters `P1`, `P2`, ..., `Pn` to the function `fun`.

The gradients of $e(k)$ with respect to the weights and biases for the hidden layer are given by:

$$\begin{aligned}\frac{\partial}{\partial b^{\mathrm{o}}} e(k) &= -1 \\ \frac{\partial}{\partial w_j^{\mathrm{o}}} e(k) &= -\tanh\Big(v_j(k)\Big)\end{aligned}$$

The gradients for the output layer are given by:

$$\begin{aligned}\frac{\partial}{\partial b_j^{\mathrm{h}}} e(k) &= -w_j^{\mathrm{o}} \frac{\partial}{\partial v_j} \tanh\Big(v_j(k)\Big) \\ &= -w_j^{\mathrm{o}}\Big(1-\tanh^2\Big(v_j(k)\Big)\Big) \\ \frac{\partial}{\partial w_{ij}^{\mathrm{h}}} e(k) &= -w_j^{\mathrm{o}} \frac{\partial}{\partial v_j} \tanh\Big(v_j(k)\Big)\phi_i(k) \\ &= -w_j^{\mathrm{o}}\Big(1-\tanh^2\Big(v_j(k)\Big)\Big)\phi_i(k)\end{aligned}$$

We write the following m-file to compute the vector e and its Jacobian in a given point:

```
function [e,J]=nnfun(x,Phi,Y,Nh)
% Objective function and Jacobian for neural network example

N=size(Y,1);      % Number of data points
Ni=size(Phi,2);   % Number of input neurons

% Extract weights and biases from x.
Wh=zeros(Ni,Nh);
Wh(:)=x(1:Ni*Nh);
Bh=zeros(1,Nh);
Bh(:)=x(Ni*Nh+1:Ni*Nh+Nh);
Wo=zeros(Nh,1);
Wo(:)=x(Ni*Nh+Nh+1:Ni*Nh+2*Nh);
Bo=x(Ni*Nh+2*Nh+1);

% Compute the output of the neural network.
V=Phi*Wh+ones(N,1)*Bh;
Ye=tanh(V)*Wo+ones(N,1)*Bo;

e=(Y-Ye);
plot([Y Ye e]); % Display the training.
drawnow;

% Compute the Jacobian if necessary.
if ( nargout > 1 )
   thV=tanh(V');
   dBo=ones(1,N);
   dWo=thV;
   dBh=(Wo*ones(1,N)).*(ones(Nh,N)-(thV).^2);
   dWh=zeros(Ni*Nh,N);
   for i=1:Nh,
      dWh(Ni*(i-1)+1:Ni*i,:)=(ones(Ni,1)*dBh(i,:)).*Phi';
   end;
   J=-[dWh; dBh; dWo; dBo]';
end;
```

Since the training of the neural network is a nonlinear optimization problem, it is very important to have good initial estimates of the parameters W^{h}, W^{o}, B^{h}, and b^{o}. The initial weights and biases for the hidden neurons should be such that the signals $v_j(k)$ excite the activation function in its linear region, i.e., around zero. If the signals $v_j(k)$ are too large, then the output of the activation functions will be either 1 or -1, and it will be very difficult to train the network. We calculate initial values W^{h} and B^{h} as follows [40]. First, the output data is scaled in the range $[-1,1]$:

$$y^s(k) = \frac{2}{y_{\max} - y_{\min}}(y(k) - y_{\min}) - 1$$

where $y_{\min}$ and $y_{\max}$ denote the minimum and maximum value of $y(k)$, respectively. Next, the scaled output data is partitioned into four parts: $y^{s,1}$, $y^{s,2}$, $y^{s,3}$, and $y^{s,4}$. Each part is used to initialize a hidden neuron by solving a linear least squares problem:

$$y^{s,j} \approx \Phi^j w_j^{\mathrm{h}} + b_j^{\mathrm{h}}, \quad j = 1,2,3,4$$

where Φ^j is the jth partitioning of the inputs to the network corresponding to $y^{s,j}$, and w_j^{h} is the jth column of W^{h}. As initial guess for b^{o} we take the mean of the output signal $y(k)$. The weights of the output layer W^{o} are initialized with random values from a Gaussian distribution with zero mean and unit variance. The following m-file `nninit.m` computes an initial guess for the parameters of the neural network:

```
function x=nninit(Phi,Y,Nh)
% Initialize the neural network.

N=size(Y,1);      % Number of data points.
Ni=size(Phi,2);   % Number of input neurons.

% Scale the output data.
ymax=max(Y);
ymin=min(Y);
Ys=2/(ymax-ymin)*(Y-ones(N,1)*ymin)-1;

% Partition the data and estimate weights and biases of the hidden
% layer.
Nb=floor(N/Nh);    % Number of samples in each partition
Wh=zeros(Ni,Nh);   % Weights
Bh=zeros(1,Nh);    % Biases
for i=1:Nh,
   WB=pinv([Phi(Nb*(i-1)+1:Nb*i,:),ones(Nb,1)])*Ys(Nb*(i-1)+1:Nb*i,:);
   Wh(:,i)=WB(1:Ni,:);
   Bh(:,i)=WB(Ni+1,:);
end;

x(1:Ni*Nh)=Wh(:);                          % Weights of hidden layer
x(Ni*Nh+1:Ni*Nh+Nh)=Bh(:);                 % Biases of hidden layer
x(Ni*Nh+Nh+1:Ni*Nh+2*Nh)=randn(Nh,1);      % Weights of output layer
x(Ni*Nh+2*Nh+1)=mean(Y);                   % Bias of output layer
```

In Figure 9.5 the output of the optimized neural network is shown. In this figure also the error $e(k)$ is shown. Clearly, the error is small compared to the output $y(k)$; the RMS of $y(k)$ amounts to 0.570, while the RMS of $e(k)$ amounts to 0.009. Hence, the neural network is able to accurately approximate the original system. □

9.4 Constrained nonlinear optimization

Consider the constrained nonlinear optimization problem

$$\min_x f(x) \qquad \text{subject to } h(x) = 0 \text{ and } g(x) \leqslant 0,$$

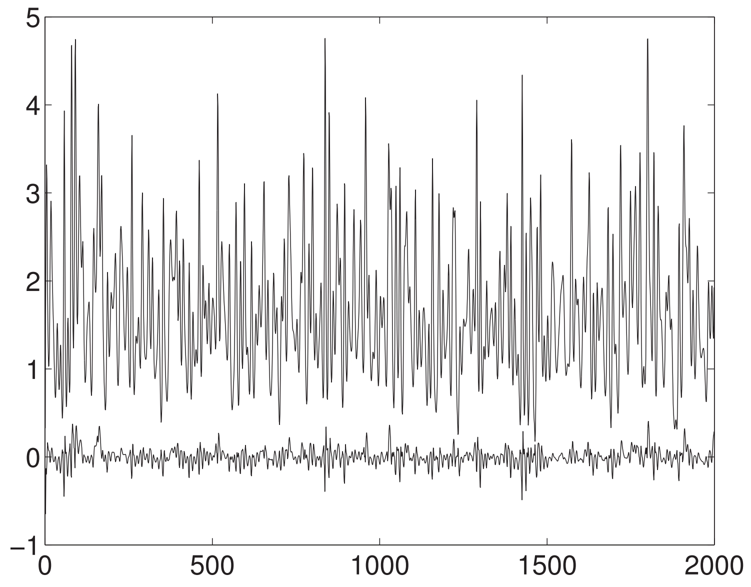


Figure 9.5: The output of the optimized neural network of Example 9.6 (top) and the error between the original output signal and the output of the network (bottom).

where one or all of the functions f, g and h are nonlinear in x. To solve this optimization problem, the function `fmincon` can be used. This function provides the Sequential Quadratic Programming method (see Section 5.2.3). The syntax of `fmincon` is:

```
x=fmincon(@fun,x0,A,b,Aeq,beq,lb,ub,@nonlcon,options)
```

where

`fun`	=	The name of an m-file describing the objective function. The function `fun` should return a scalar function value `f` in the given point `x` as its first output argument. In this case the calling syntax of `fun` is `f=fun(x)`. If the gradient of `fun` can also be computed and the option `GradObj` is set to `on`, then the function `fun` must return the gradient vector `g` as second output argument. The calling syntax of `fun` is then `[f,g]=fun(x)`. The gradient must be provided to use the large-scale method. It is optional for the medium-scale method (if it is not provided by the user a finite-difference approximation will be used).
`x0`	=	Initial starting point.
`A, b`	=	Coefficients of the inequality constraints (`A x` $\leqslant$ `b`).
`Aeq, beq`	=	Coefficients of the equality constraints (`Aeq x = beq`). These arguments are optional. Set `Aeq=[]; beq=[];` if no equality constraints exist.
`lb, ub`	=	Lower and upper bounds for `x`, i.e., `lb` $\leqslant$ `x` $\leqslant$ `ub`. These arguments are optional. By default no bounds are imposed.

`nonlcon` =	The name of an m-file describing the nonlinear inequality constraints `c(x)<=0` and the nonlinear equality constraints `ceq(x)=0`. The function `nonlcon` accepts a vector `x` and returns two vectors `c` and `ceq`, containing respectively the nonlinear inequalities and equalities evaluated at `x`. In this case the calling syntax of `nonlcon` is `[c,ceq]=nonlcon(x)`. If the gradients of the constraints can also be computed and the option `GradConstr` is set to `on`, then the function `nonlcon` must return the gradient vector matrices `Jc` and `Jceq` as third and fourth output argument. The *j*th column of the matrix `Jc` is the gradient of the *j*th inequality constraint[3]; `Jceq` is defined similarly. The calling syntax is then `[c,ceq,Jc,Jceq]=nonlcon(x)`.
`options` =	Options. This argument is optional.

The most important options for `fmincon` are:

`Algorithm`	: Selects the algorithm to use. Possible values are `trust-region-reflective` (default; note that this is a large-scale algorithm), `active-set`, `interior-point`, and `sqp`.
`DerivativeCheck`	: If set to `on`, then the user-supplied derivatives (gradients) will be compared to finite differencing derivatives. The default value is `off`.
`Diagnostics`	: Prints diagnostic information about the problem to be solved if set to `on`. The default value is `off`.
`Display`	: Controls the display of (intermediate) values. Possible values are: `off` (no output), `iter` (displays output at each iteration), `notify` (displays output only if `fmincon` does not converge), and `final` (displays just the final output). The default value is `final`.
`GradObj`	: Indicates whether the gradient of the objective function is defined by the user.
`GradConstr`	: Indicates whether the gradients of the constraints are defined by the user.
`MaxFunEvals`	: Maximum number of function evaluations allowed (default value: 100 $\times$ number of variables, except for the interior-point algorithm, for which the default value is 3000).
`MaxIter`	: Maximum number of iterations allowed (default value: 400, except for the interior-point algorithm, for which the default value is 1000).
`TolCon`	: Termination tolerance on the constraint violation (default value: 10^{-6}).
`TolFun`	: Termination tolerance on the function value (default value: 10^{-6}).
`TolX`	: Termination tolerance on `x` (default value: 10^{-6}).

It is also possible to use extra output arguments to obtain more information about the optimization process. The command

```
[x,fval,exitflag,output]=fmincon(@fun,x0,A,b)
```

also returns the final value of the objective function `fval`, and a flag variable `exitflag` describing the exit condition (>0 if the algorithm converged to a solution `x`, 0 if the maximum number of function evaluations or iterations was exceeded, and <0 if the algorithm did not converge

[3]Hence, `Jc` is the *transpose* of the Jacobian of `c`, and `Jceq` is the transpose of the Jacobian of `ceq`.

to a solution). The structure `output` contains among others the number of function evaluations (`output.funcCount`), and the number of iterations (`output.iterations`).

Example 9.7 Consider the nonlinear objective function

$$f(x_1,x_2,x_3) = -x_1 x_2 e^{x_3}$$

The objective is to minimize this function subject to the constraints

$$x_1 + 2x_2 \leqslant 2 \tag{9.1}$$

$$x_1^2 + 2x_2^2 + 3x_3^2 \leqslant 9 \tag{9.2}$$

First, we write m-files `fmincon_example_fun.m` and `fmincon_example_nonlcon.m` that describe the objective function as well as the nonlinear constraint (9.2):

```
function f=fmincon_example_fun(x)
f=-x(1)*x(2)*exp(x(3));

function [c,ceq]=fmincon_example_nonlcon(x)
c=x(1)^2+2*x(2)^2+3*x(3)^2-9;
ceq=[];
```

Next, we define the linear inequality constraint (9.1), some initial guess, set some options and invoke the function `fmincon`.

```
A=[1 2 0];
b=2;
x0=[1 1 1]';
% Select the SQP algorithm and display intermediate results.
options=optimoptins('fmincon',...
   'Algorithm','sqp','Display','iter');
x=fmincon(@fmincon_example_fun,x0,A,b,[],[],[],[],...
          @fmincon_example_nonlcon,options)
```

Let us now repeat this example, but with the gradient of the objective function and the Jacobian of the constraints provided by the user. We write m-files `fmincon_example_fun_2.m` and `fmincon_example_nonlcon_2.m` that describe the objective function and its gradient, as well as the nonlinear constraint and its (transposed) Jacobian:

```
function [f,g]=fmincon_example_fun_2(x)
f=-x(1)*x(2)*exp(x(3));
%
% If the function is called with 2 output arguments then
% also compute the gradient.
%
if ( nargout > 1 )
   g=[-x(2)*exp(x(3))
      -x(1)*exp(x(3))
      -x(1)*x(2)*exp(x(3))];
```

```
end;

function [c,ceq,Jc,Jceq]=fmincon_example_nonlcon_2(x)
c=x(1)^2+2*x(2)^2+3*x(3)^2-9;
ceq=[];
%
% If the function is called with 3 or 4 output arguments then
% also compute the transposed Jacobians.
%
if ( nargout > 2 )
   Jc=[2*x(1)
       4*x(2)
       6*x(3)];
   Jceq=[];
end;
```

Next, we define the linear inequality constraint (9.1), some initial guess, set some options and invoke the function `fmincon`.

```
A=[1 2 0];
b=2;
x0=[1 1 1]';
% Select the SQP algorithm and display intermediate results.
options=optimoptions('fmincon','Algorithm','sqp',...
   'Display','iter',...
   'GradObj','on','GradConstr','on');
x=fmincon(@fmincon_example_fun_2,x0,A,b,[],[],[],[],...
          @fmincon_example_nonlcon_2,options)
```

With the gradient and the Jacobian specified by the user, the algorithm terminates in 7 iterations, 12 function evaluations, and 0.0145 s CPU time[4], whereas the algorithm with the gradient and the Jacobian computed by a finite-difference method terminates in 7 iterations, 36 function evaluations, and 0.0161 s CPU time. □

Another example of the use of `fmincon` can be found in Chapter 14, where it is used to solve a multi-criteria controller design problem.

Remark 9.8 The Optimization Toolbox also contains the functions `fgoalattain` for multi-criteria optimization (cf. Chapter 10), and `bintprog` for linear programming problems with binary variables only (cf. Chapter 11).
The functions `ga` and `simulannealbnd` of the Global Optimization Toolbox implement respectively a genetic algorithm and simulated annealing. ◇

[4]On a 2.66 GHz Intel Core 2 Quad Q8400 CPU.

Chapter 10

Multi-Objective Optimization

In this chapter we consider an optimization problem in which several, often competing, objectives have to be optimized. This chapter is primarily based on [42].

10.1 Problem statement

The rigidity of the mathematical problem posed by the general optimization formulation

$$\begin{aligned}&\min_x f(x)\\&\text{s.t. } g(x) \leqslant 0 \text{ and } h(x) = 0 \ ,\end{aligned}$$

which has been considered in the previous chapters, is often remote from that of a practical design problem. Rarely does a single objective with several hard constraints adequately represent the problem being faced. More often, there is a vector of objectives that must be traded off in some way. The relative importance of these objectives is not generally known until the system's best capabilities are determined and trade-offs between the objectives fully understood. As the number of objectives increases, trade-offs are likely to become complex and less easily quantified. There is much reliance on the intuition of the designer, and his or her ability to express preferences throughout the optimization cycle. Thus, important requirements for a multi-objective design strategy are that it should enable us to put forward a natural problem formulation, and yet it should also allow us to solve the problem and to enter preferences into a numerically tractable and realistic design problem.

Multi-objective optimization is concerned with the minimization of a vector of objectives $F(x) \in \mathbb{R}^m$ that can be the subject of a number of constraints or bounds:

$$\min_{x \in \mathbb{R}^n} F(x) \tag{10.1}$$

$$\text{s.t. } g(x) \leqslant 0 \text{ and } h(x) = 0 \ . \tag{10.2}$$

Note that because $F(x)$ is a vector, if any of the components of $F(x)$ are competing, there is no unique solution to this problem. Instead, the concept of non-inferiority (also called Pareto optimality) must be used to characterize the objectives.

10.2 Pareto optimality

A *Pareto optimal point* x^* has the property that there does not exist another feasible point $\tilde{x}$ such that $F(\tilde{x}) \leqslant F(x^*)$ and $F_i(\tilde{x}) < F_i(x^*)$ for some i. Consider, e.g., Figure 10.1 where we have represented the

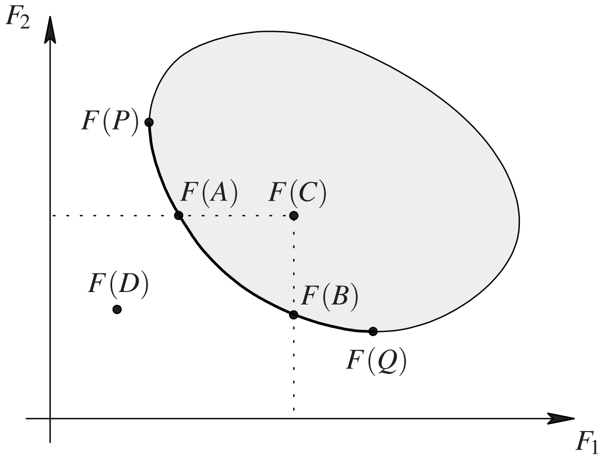


Figure 10.1: Illustration of Pareto optimality. All the points on the part of the boundary of the feasible region that is indicated by the full line are Pareto optimal points.

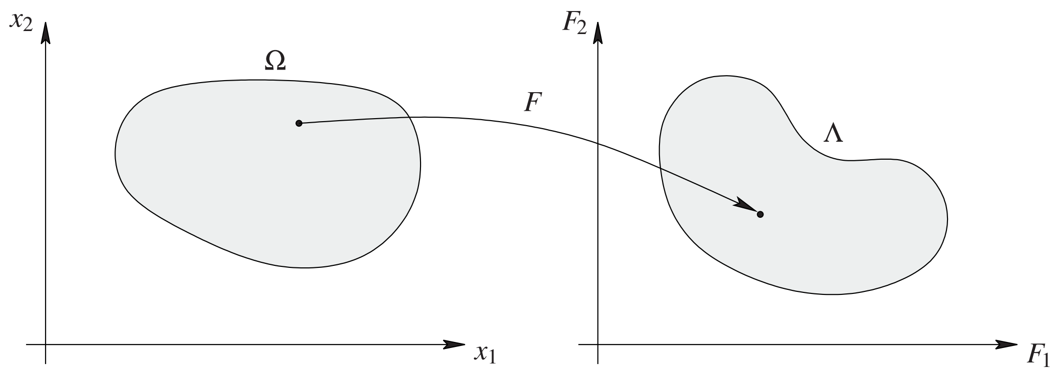


Figure 10.2: Mapping from parameter space into objective function.

functions values $F(x) = [\,F_1(x)\;\; F_2(x)\,]^T \in \mathbb{R}^2$ for some points $x = A,B,C,D,P,Q$. The shaded region corresponds to the function values of the feasible points. Clearly, A, B and C are feasible, whereas D is infeasible. Note that C is not Pareto optimal since $F(A) \leqslant F(C)$ and $F_1(A) < F_1(C)$.

However, A is a Pareto optimal point since there does not exist any feasible solution x such that $F(x) \leqslant F(A)$ and $F_1(x) < F_1(A)$ or $F_2(x) < F_2(A)$, i.e., if starting at A we want to keep a feasible solution while decreasing F_1 (or F_2), some other component of the performance vector F has to increase. The point B is also a Pareto optimal point. In fact, each point on the part of the boundary of the feasible region that is indicated by the thick curved line between $F(P)$ and $F(Q)$ is a Pareto optimal point.

To define the concept of Pareto optimality more precisely, consider the feasible region Ω in the parameter space, i.e.,

$$\Omega = \{\, x \in \mathbb{R}^n \mid g(x) \leqslant 0 \text{ and } h(x) = 0 \,\} \;. \tag{10.3}$$

This allows us to define the corresponding feasible region Λ for the objective function space as follows:

$$\Lambda = \{\, y \in \mathbb{R}^m \mid y = F(x) \text{ for some } x \in \Omega \,\} \;. \tag{10.4}$$

So the performance function F maps the parameter space into the objective function space as is represented for a two-dimensional case in Figure 10.2. A Pareto optimal point can now be formally defined as follows.

Definition 10.1 (Pareto optimal point) *Consider the multi-objective problem (10.1)–(10.2) and the set Ω defined in (10.3). A point $x^* \in \Omega$ is a Pareto optimal point (or a non-inferior solution) if for some neighborhood $\mathcal{N}_{x^*}$ of x^* there does not exist a point $\tilde{x} \in \mathcal{N}_{x^*}$ such that $\tilde{x} \in \Omega$ and*

$$\begin{aligned} &F(\tilde{x}) \leqslant F(x^*) \\ &F_i(\tilde{x}) < F_i(x^*) \quad \textit{for some } i. \end{aligned}$$

10.3 Solution methods for multi-objective optimization problems

Since any point in Ω that is not a Pareto optimal point represents a point in which improvement can be obtained in one or more of the objectives, it is clear that such a point is of limited value. Therefore, multi-objective optimization is concerned with the generation and selection of Pareto optimal points. The techniques for multi-objective optimization are wide and varied and all the methods cannot be covered within the scope of this chapter. In the next subsections we will describe some of the techniques, in particular:

- the weighted sum strategy,
- the ε-constraint method,
- the goal attainment method.

10.3.1 Weighted sum strategy

The weighted sum strategy converts the multi-objective problem of minimizing the vector F into a scalar problem by constructing a weighted sum of all the objectives (with nonnegative weights). This results in the following optimization problem:

$$\min_{x \in \Omega} \sum_{i=1}^{m} w_i F_i(x) \tag{10.5}$$

with $w_i \geqslant 0$ for $i = 1, \ldots, m$. The problem can then be optimized using a standard constrained optimization algorithm (see Chapter 5).

The main problem of the weighted sum strategy consists in assigning appropriate weighting coefficients to each of the objectives. The weighting coefficients do not necessarily correspond directly to the relative importance of the objectives or allow trade-offs between the objectives to be expressed. Furthermore, if the region Λ is non-convex, then not all Pareto optimal solutions are accessible by an appropriate selection of the weights w_i. This can be illustrated geometrically as follows. Consider the two-objective case in Figure 10.3. For a given real number c we can draw a line L_c in the objective function space defined by the points for which $w^T F(x) = c$. The minimization of the objective function of problem (10.5) can be interpreted as finding the smallest value of c for which L_c just touches the boundary of Λ (recall the graphical interpretation of linear programming considered in Chapter 2). Note that the selection of the weights w defines the slope of L_c, which in turn leads to the solution point where L_c touches the boundary of Λ.

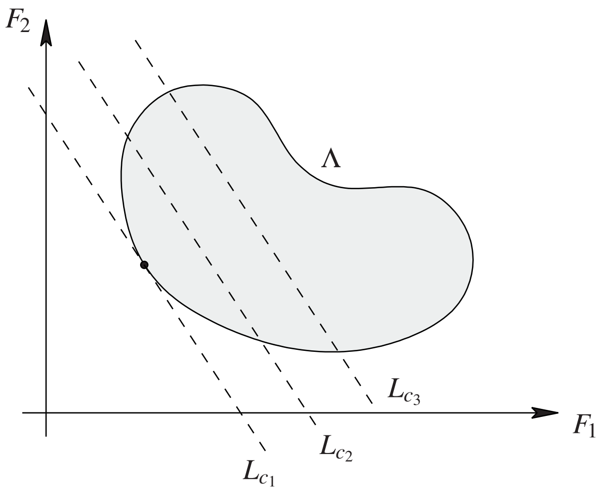


Figure 10.3: Geometrical representation of the weighted sum method.

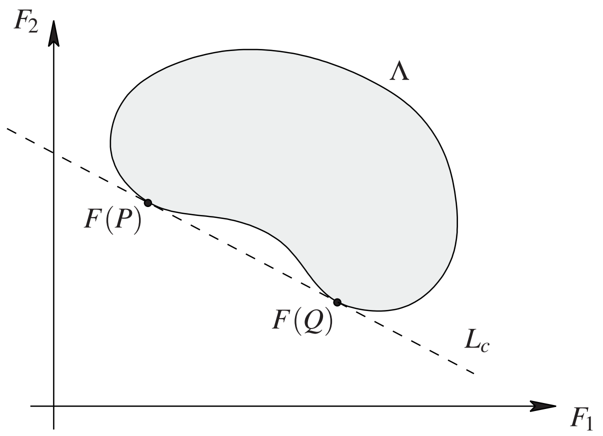


Figure 10.4: The weighted sum method for a non-convex solution boundary.

The problem that not all Pareto optimal points can be obtained by selecting an appropriate value for w arises when the lower boundary of Λ is non-convex as shown in Figure 10.4. In this case the set of Pareto optimal points between $F(P)$ and $F(Q)$ is not available.

10.3.2 ε-constraint method

A procedure that overcomes some of the convexity problems of the weighted sum technique is the ε-constraint method. This involves minimizing a primary objective, F_{i_p}, and expressing the other objectives in the form of inequality constraints:

$$\begin{aligned}
&\min_{x\in\Omega} F_{i_\mathrm{p}}(x) \\
&\text{s.t. } F_i(x) \leqslant \varepsilon_i \quad \text{for } i = 1,\ldots,m \text{ with } i \neq i_\mathrm{p}\,.
\end{aligned}$$

Note that this optimization problem can also be solved using one of the constrained optimization methods discussed in Chapter 5. Figure 10.5 shows a two-dimensional representation of the ε-constraint

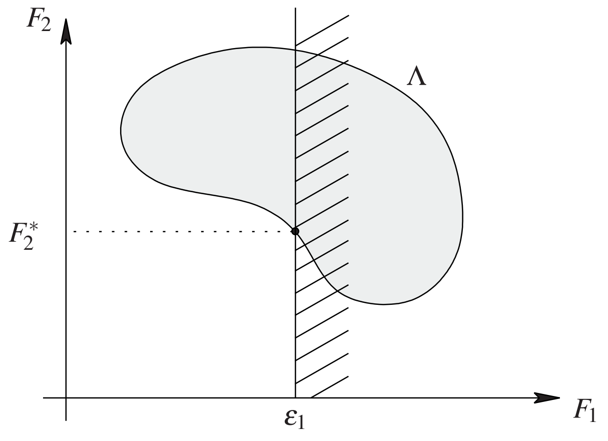


Figure 10.5: Geometrical representation of the ε-constraint method.

method for a two-objective problem of the form

$$\min_{x\in\Omega} F_2(x) \quad \text{s.t. } F_1(x) \leqslant \varepsilon_1.$$

The ε-constraint method is able to identify a number of Pareto optimal solutions on a non-convex boundary that are not obtainable using the weighted sum technique such as, e.g., the solution point $(F_1, F_2) = (\varepsilon_1, F_2^*)$ in Figure 10.5. A problem with this method is, however, a suitable selection of ε to ensure a feasible solution. A further disadvantage of this approach is that the use of hard constraints is rarely adequate for expressing true design objectives. Although there exist methods that, e.g., prioritize the objectives and allow the optimization to proceed with reference to these priorities and allowable bounds of acceptance, the difficulty there is in expressing such information at early stages of the optimization cycle.

In order for the designers' true preferences to be put into a mathematical description, the designers must express a full table of their preferences and satisfaction levels for a range of objective value combinations. A procedure must then be realized that is able to find a solution with reference to these data. Such methods have been derived, but they are deemed impractical for most practical design problems.

What is required is a formulation that is simple to express, retains the designers' preferences, and is numerically tractable. The goal attainment method aims at fulfilling these requirements.

10.3.3 Goal attainment method

In this method we first define a vector of design goals, $F^* = [F_1^* \; F_2^* \; \ldots \; F_m^*]^T$, that is associated with the vector of objectives $F(x) = [F_1(x) \; F_2(x) \; \ldots \; F_m(x)]^T$. The problem formulation allows the objectives to be under- or overachieved, enabling the designer to be relatively imprecise about initial design goals. The relative degree of under- or over-achievement of the goals is controlled by a vector w of weighting coefficients, and is expressed as a standard optimization problem using the following formulation:

$$\min_{\gamma\in\mathbb{R},\, x\in\Omega} \gamma \tag{10.6}$$

$$\text{s.t. } F_i(x) - w_i\gamma \leqslant F_i^* \quad \text{for } i = 1, \ldots, m. \tag{10.7}$$

The term $w_i\gamma$ in the inequality constraints (10.7) introduces an element of slackness into the problem, which otherwise imposes that the goals be rigidly met. The weighting vector w enables the designer to express a measure of the relative trade-offs between the objectives. For example, setting the weighting vector w equal to the initial goals indicates that the same percentage under- or over-attainment of the goals F^* is achieved. One can incorporate hard constraints into the design by setting a particular weighting factor to zero.

The goal attainment method provides a convenient intuitive interpretation of the design problem, which is solvable using standard optimization procedures. In particular, the problem (10.6)–(10.7) is a nonlinear constrained optimization problem that can, e.g., be solved using sequential quadratic programming (cf. Chapter 5).

Chapter 11

Integer Optimization[*]

In this chapter we consider optimization problems with both real-valued and integer-valued parameters:

$$\min_{x} f(x) \tag{11.1}$$

$$\text{subject to } h(x) = 0 \tag{11.2}$$

$$g(x) \leqslant 0 \tag{11.3}$$

$$x = \begin{bmatrix} x_\text{r} \\ x_\text{i} \end{bmatrix} \tag{11.4}$$

$$x_\text{r} \in \mathbb{R}^{n_\text{r}},\ x_\text{i} \in \mathbb{Z}^{n_\text{i}} \ . \tag{11.5}$$

If x consists of both real-valued and integer-valued parameters, the problem is called a *mixed integer optimization problem*; a problem with only integer valued parameters is called an *all-integer optimization problem*.

The goal of the optimization is to find a solution which is feasible and optimal. However, as will be shown later, finding the global optimum is, in many cases, impossible in practice (this will be motivated in Section 11.1). In practice, the goal is then reduced to finding a feasible solution which preferably approaches the optimality of the global optimum.

Note that problem (11.1)–(11.5) can be rewritten as

$$\min_{\alpha,x} \ \alpha \tag{11.6}$$

$$\text{subject to } h(x) = 0 \tag{11.7}$$

$$f(x) - \alpha \leqslant 0 \tag{11.8}$$

$$g(x) \leqslant 0 \tag{11.9}$$

$$x = \begin{bmatrix} x_\text{r} \\ x_\text{i} \end{bmatrix} \tag{11.10}$$

$$x_\text{r} \in \mathbb{R}^{n_\text{r}},\ x_\text{i} \in \mathbb{Z}^{n_\text{i}},\ \ \alpha \in \mathbb{R} \ . \tag{11.11}$$

So by finding the smallest α for which there exists a feasible solution, problem (11.1)–(11.5) can be solved. This means that also algorithms that only search for feasible solutions, can be used as a basic tool in an optimization procedure, if we use them in combination with, e.g., a binary search procedure for α in order to determine the smallest value of α for which the system (11.7)–(11.11) is still feasible.

[*]Parts of this chapter are taken from the PhD thesis of Victor Terpstra [41].

Integer parameters may represent various quantities like the number of light bulbs produced by a factory per year, the number of students that pass the "Optimization in Systems and Control" examination, or the position of a switch in an electric circuit. In the case of the light bulbs, the approximation by a real value will probably cause negligible errors. Depending on the number of students that pass the examination, the integer number of students may or may not be approximated by a real value. In the case of a binary switch, the value cannot be approximated properly by a real value, and a different optimization strategy has to be applied. It is therefore important to consider on beforehand whether an integer optimization is really necessary, or whether a real-valued optimization performs satisfactorily.

11.1 Complexity

The main problem in integer optimization is the complexity. Many integer optimization problems are mathematically classified as NP-hard problems. In practice, this means that the number of potential solutions (the search space) grows exponentially as a function of the problem size. The simplest approach to solving the optimization problem is to evaluate all potential solutions. Because of the exponential growth of the solution space, this will require an exponentially growing calculation time.

A logical question in these times of cheap and super-fast computers is:

"SO WHAT!?"

"If the current computer is not able to solve the problem in a reasonable time, just buy the latest model or wait for next year's models when processor speeds will have doubled or quadrupled. Or, to go along with the latest trends, buy a parallel machine and just install enough processors."

Unfortunately, this approach does not solve the problem of the combinatorial explosion. Let us illustrate this by considering the following scheduling problem. A plant consists of 10 units; batches have to be scheduled over a horizon of 5 days. A discretized time slot approach has been chosen with a one-hour grid. This would result in 120 binary variables per batch and per unit, each indicating whether a certain batch uses a certain unit during a certain hour. For one batch, this results in 1200 binary variables, theoretically leading to $2^{1200} = 10^{308}$ potential solutions. Thus 10 batches would lead to $2^{12000} = 10^{3612}$ potential solutions.
Again, one could ask:

"SO WHAT!?"

In order to give an idea of how big this number is, consider an estimation of the upper limit of the capacity of the biggest and fastest computer which could ever be built. In summary, the following assumptions have been made. A processor will never become smaller than a proton (i.e., about 10^{-15} meter). The number of processors which a parallel computer may contain will always be smaller than the volume of the entire universe divided by the volume of a single proton (i.e., about 10^{126} processors). The clock period of each processor will be longer than the time light requires to cross the diameter of a proton ($3 \cdot 10^{-24}$ seconds). A processor needs at least this time to evaluate a single solution. The processors communicate at less than infinite speed (leading to $< 10^{23}$ tests per second per processor). The time available for one optimization run will always be less than 10 times the current age of the universe (leading to $< 10^{19}$ seconds). Conclusion: during this time, maximally 10^{168} solutions can be evaluated. This number is an absolute limit, or, in other words,

"There is not enough time or space to do more than 10^{168} of anything."

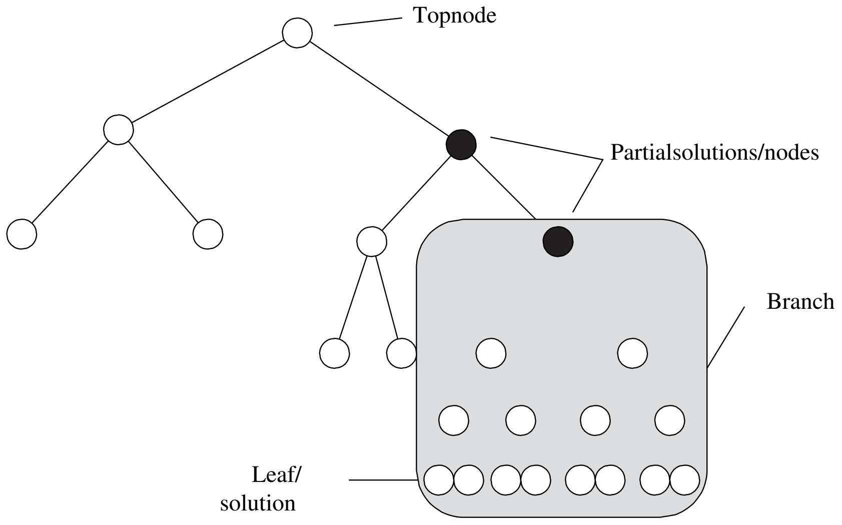


Figure 11.1: Some definitions in a search tree.

Compare this with the 10^{3612} number of solutions in the above-mentioned simple scheduling problem. This shows that enumeration even fails with small-sized problems. The only alternative is to be smart and lucky. “Smart” in this context means that a more advanced search method should be used than generate-and-test. “Lucky” means that your problem is either small enough to complete a (smart) search, or, if not, that you will find acceptable solutions with little search time. They need to be in balance with each other: the more unlucky you are, the smarter you have to be.

11.2 Search

The essence of combinatorial optimization is searching through a tree. The optimization process starts at the top node where no decisions are made. Each decision (i.e., a choice of a value for a specific parameter) that can be made at this node results in an arc leading to a child node which represents the state of the search/optimization process. Again, at these nodes new decisions can be made with equivalent results. The tree ends at child nodes called leaves if no additional decisions can be made. Each leaf represents a potential solution. The structure of the tree illustrates the exponential number of leaves as a function of the number of decisions.

Different optimization techniques can be categorized by the way they (do or do not) implement the following basic tasks in tree searching:

1. The search strategy.

2. The test for the feasibility and optimality of leaves (fully determined solutions).

Theoretically, these two are sufficient to perform a correct and complete search. In order to improve the speed of the search, four basic refinements can be recognized:

3. Preclusion / branch pruning.

4. Fusion / branch merging.

5. Search rearrangement.

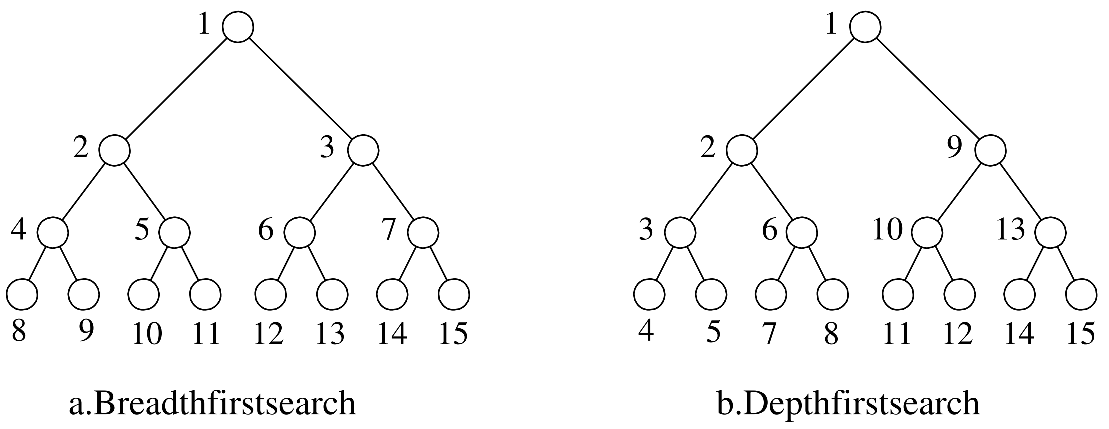


Figure 11.2: The order in which nodes are generated by: a) breadth-first, b) depth-first.

6. Problem decomposition.

The search strategy

The search strategy determines the way the search tree is constructed, i.e., via depth-first, breadth-first, a combination of these, or via direct generation of leaves.

Most integer-optimization techniques generate a single candidate solution step by step, which is in fact a depth-first search. Breadth-first would mean it would generate all candidate solutions step-by-step simultaneously. It is rarely used because it requires large amounts of memory and it always takes a long time before a feasible solution is found. Combinations of breadth-first and depth-first are also possible.

Another commonly used technique is to generate a fully determined candidate solution in one step. This means that only leaves are generated and tested. This is, in fact, what most NLP solvers do (e.g., hill-climbers and stochastic search techniques).

In dynamic optimization there are two approaches: the sequential and the simultaneous approach.

- In the sequential approach, the optimization consists of two separate, iterating modules: a selection/optimization module which generates fully determined solution candidates and an evaluation module which evaluates these candidate solutions, after which the selection module generates a new candidate, and so on. Both are independent modules, which do not need to know anything about each other, nor can they use each other's internal information.

- In the simultaneous approach, the information obtained while testing (partial) solutions is used for the selection of new (partial) solutions. Therefore, both the techniques are closely coupled in one module.

In terms of tree searching, this difference can be viewed as the sequential approach, only generating leaves in one step, where the simultaneous approaches are able to use search strategies that use partial solutions.

The test for feasibility and optimality of leaves

This is the most basic and necessary feature of any optimization technique. It is equivalent to the simulation of a solution.

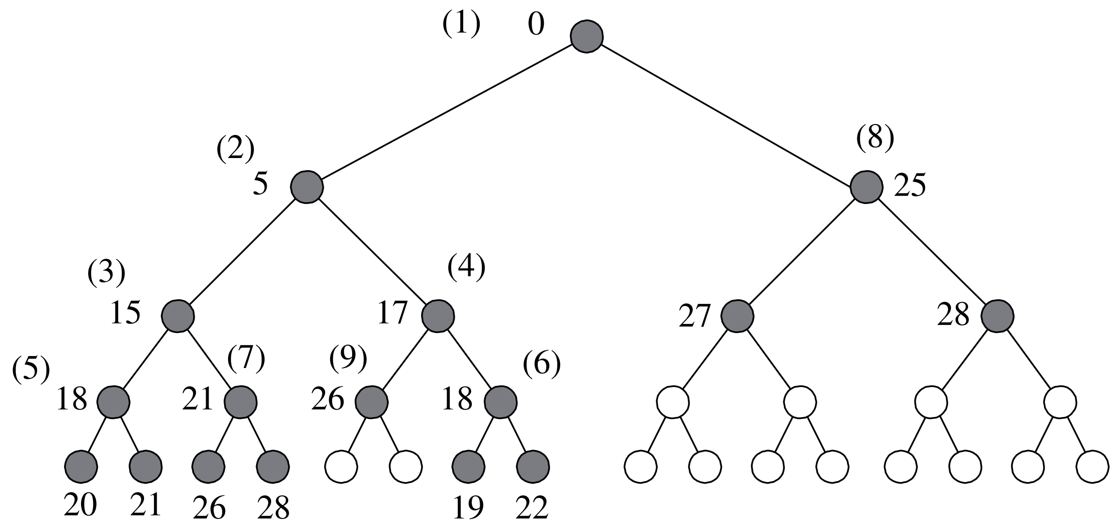


Figure 11.3: The order in which nodes would be generated by a simultaneous search. The number between the brackets indicates the sequence in which the nodes are expanded. A simultaneous search can use the same cost estimation as needed for branch-and-bound.

Preclusion / branch pruning

Preclusion is obtained by a test for the feasibility and optimality of nodes (i.e., partial solutions). If a partial solution is rejected, none of its descendants need to be evaluated. Therefore, a single test (at a node) can replace many tests (at all the descendants of the node). Only in cases for which such a test exists, will depth-first (and breadth-first) techniques be more useful than direct generation of leaves. This type of test itself contains three basic tests:

1. Test of feasibility of the decisions made so far. An example of such a test in the scheduling of a batch process that consists of three batches, could be the following: in a partial solution in which the first of the three batches has to be scheduled, the test examines only the schedule for this one batch (e.g., whether it satisfies its deadline).

2. Test of feasibility of the descendants of this partial solution. In the above example, the schedule of the first batch might be feasible, but it might be possible to see that (given this partial solution) the two other batches can never be scheduled without violating any constraints. One could say that this type of test "looks ahead" downwards in the tree.

3. Evaluation of the optimality of a partial solution. The evaluation calculates a lower bound of the cost function of each descendant leaf of this node. This lower bound is used in combination with a branch-and-bound technique: if the lower bound is higher than the costs of a previously found solution, it proves that all descendant leaves will be worse than in the previously found solution and therefore need not be evaluated.

Fusion / branch merging

The idea of branch merging is to avoid doing the same thing twice: if two or more subtrees of the tree are isomorphic, it is only necessary to search one of them. For example, if the shortest way from A to

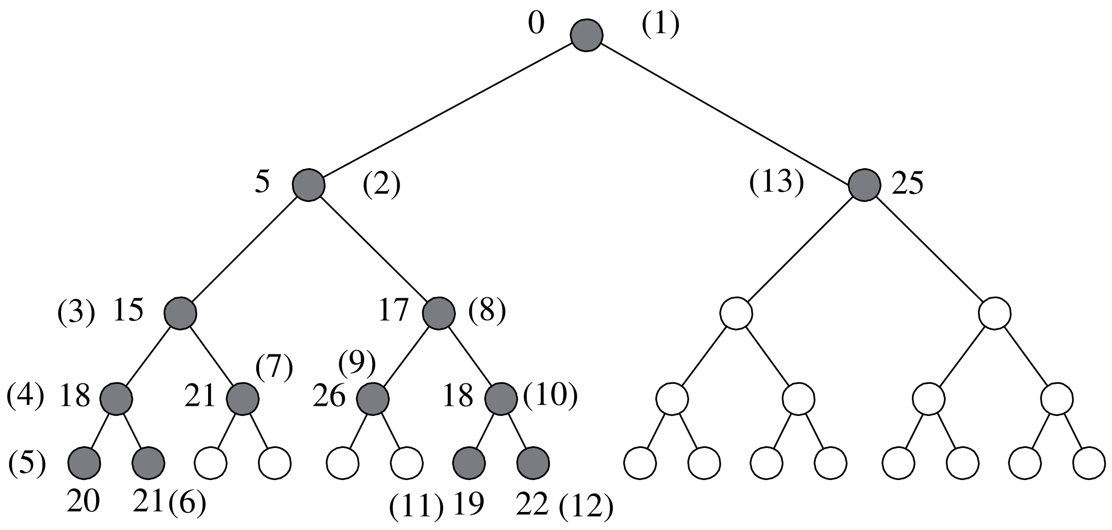


Figure 11.4: Illustration of a search using branch-and-bound. In the first depth-first search a solution is found with cost 20. So f^* is set to 20. While backtracking, other partial solutions are evaluated. If the lower bound on the costs of a partial solution is higher than f^*, there is no need to evaluate any other node of this branch (only the shaded nodes have been evaluated). If during the search a better solution is found, then f^* will be updated.

B is required, and we found two ways to come to a point C halfway, then the "subtree" from C to B only has to be evaluated once to obtain its partial contribution.

Search rearrangement

Search rearrangement is also called Value and Variable Ordering (VVO). When at a node the optimizer has the opportunity to choose from more than one alternative, without knowing which one is the correct one (which, in general, will be the case), it has to decide which one it will evaluate first. First, it has to select the variable and then the value it will assign to it. A VVO is based on heuristics which state a preference for a choice above other choices. By definition, a heuristic cannot select the right choice for certain. However, on average, using heuristics is supposed to give better results than performing a random selection among the potential choices. A good VVO is important for two reasons:

1. For real-time applications. In real-time optimization it is important to find a feasible and reasonably good solution as fast as possible. When an optimal solution is required before the optimizer has finished the search (i.e., has found the optimal solution), it can use this best feasible solution obtained so far.

2. For branch-and-bound efficiency. The better the first-found solutions are, the better the branch-and-bound will work. Recall that the branch-and-bound method allows us to prune branches at nodes in the tree. At higher-level nodes, which are closer to the root of the tree, the branches are longer and wider and therefore pruning them will result in greater reduction of the search space.

Problem decomposition

Problem decomposition means that the full problem is decomposed into a number of subproblems. Each of the subproblems is solved separately. The solutions to the subproblems are merged into one solution of the original problem.

An important condition for the application of this technique is, of course, that the problem allows decomposition. This can be at two levels: feasibility and optimality. It is necessary that feasible solutions of all subproblems lead to a composed solution that is also feasible. The problem decomposition would be perfect if additionally the optimal solutions of all subproblems would also lead to a composed solution that is optimal for the original problem.

11.3 Overview of integer optimization methods

There is a range of approaches to tackling integer optimization problems. Some are already being used in practice and others are new developments. The most important approaches are:

- MI(N)LP (Mixed Integer (Non)-Linear Programming).
- Dynamic programming
- CLP (Constraint Logic programming)
- Heuristic search

11.3.1 Mixed Integer (Non-)Linear Programming (MI(N)LP)

MILP

A basic solution method for MILP is based on a depth-first branch-and-bound procedure. In each node, the MILP formulation is relaxed to an LP formulation by defining the integer values as continuous. This LP model is optimized. The value of the criterion function is used as a lower bound by which the branch-and-bound cuts the branches. The real-valued value of an integer variable is used to decide on the branching. For example, for an integer variable $x \in \mathbb{Z}$, the LP optimum is $x = 4.7$. Then two branches are created: one with the constraint $x \leqslant 4$ and the other with the constraint $x \geqslant 5$.

The advantage of the MILP is that it is a clear, widespread and very generic type of modeling for which ready-to-use solvers exists. The LP solvers in particular have been explored and optimized for performance (see Chapter 2). A disadvantage is that for binary values the use of MILP is not reasonable.

MINLP

The extension to MINLP is straightforward, only instead of the LP optimization a nonlinear optimization has to be done. If the (real-valued) nonlinear optimization function is a convex one, the MINLP algorithm will converge to the global optimum. In the non-convex case a (mostly) cumbersome nonlinear optimization has to be done.

The main disadvantage of generic MI(N)LP solvers is that they are "black boxes", which cannot be influenced or interacted with, and that they are only able to solve problems of a very limited size.

Example 11.1 Consider the brewery scheduling problem of Chapter 2 where a small brewery sells draft or dark beer and where “profit” must be maximized. The problem was described by

$$\max_{x_1,x_2} \; 20x_1 + 30x_2$$

subject to

$$\begin{array}{rcrcl} x_1 & + & x_2 & \leqslant & 100 \\ 0.1x_1 & + & 0.2x_2 & \leqslant & 14 \\ & & x_1, x_2 & \geqslant & 0 \\ & & x_1, x_2 & \in & \mathbb{R} \end{array}$$

where x_1 and x_2 are the number of boxes with draft beer and dark beer, respectively. In fact, when defining this problem in Chapter 2 we should already have imposed integer constraints on x_1 and x_2. However, the optimal solution found in Chapter 2 resulted in a integer solution. So imposing an integer constraint on x_1 and x_2 will not change the solution.
An extra problem is now taken into account: The boxes of beer are transported using pallets of 16 boxes, and so the amount of produced beer should be expressed in number of pallets.
By choosing $x_1 = 16 \cdot n_1$ and $x_2 = 16 \cdot n_2$ where $n_1, n_2 \in \mathbb{Z}$, we obtain the following integer optimization problem:

$$\begin{aligned} &\max_{n_1,n_2} \; 320n_1 + 480n_2 \\ &\text{subject to } \; 16n_1 + 16n_2 \leqslant 100 \\ &\qquad\qquad\quad 1.6n_1 + 3.2n_2 \leqslant 14 \\ &\qquad\qquad\quad n_1, n_2 \geqslant 0 \\ &\qquad\qquad\quad n_1, n_2 \in \mathbb{Z} \;, \end{aligned}$$

where n_1 and n_2 denote the number of pallets of respectively draft and dark beer. The MILP algorithm now runs as follows: First choose n_1 and n_2 as real variables. Then the optimum is reached for $n_1^* = x_1^*/16 = 60/16 = 3.75$ and $n_2^* = x_2^*/16 = 40/16 = 2.5$. Now we split the problem into two subproblems: In the first subproblem we introduce an extra constraint $n_1 \leqslant 3$ and in the second subproblem we introduce and extra constraint $n_1 \geqslant 4$. In both problems n_2 is still chosen as a real variable. This leads to two subproblems SP1 and SP2:

SP1 $(n_1 \leqslant 3)$ results in $n_1^* = 3$ and $n_2^* = 46/16 = 2.875,\; f^* = 2340$

SP2 $(n_1 \geqslant 4)$ results in $n_1^* = 4$ and $n_2^* = 36/16 = 2.25,\; f^* = 2360$

In the same way we introduce the constraints for the second variable, $n_2 \leqslant 2$ and $n_2 \geqslant 3$ for both subproblem SP1 and SP2. This leads to:

SP11 $(n_1 \leqslant 3, n_2 \leqslant 2)$ results in $n_1^* = 3$ and $n_2^* = 2,\; f^* = 1920$

SP12 $(n_1 \leqslant 3, n_2 \geqslant 3)$ results in $n_1^* = 44/16 = 2.75$ and $n_2^* = 3,\; f^* = 2320$

SP21 $(n_1 \geqslant 4, n_2 \leqslant 2)$ results in $n_1^* = 4$ and $n_2^* = 2,\; f^* = 2240$

SP22 $(n_1 \geqslant 4, n_2 \geqslant 3)$ has no feasible solutions.

Note that only subproblem SP12 yields a non-integer-valued optimal solution. Therefore, the next step consists in branching subproblem SP12 into two new subproblems, the first with constraint $n_1 \leqslant 2$ and the second with $n_1 = 3$:

SP121 $(n_1 \leqslant 2, n_2 \geqslant 3)$ results in $n_1^* = 2$ and $n_2^* = 54/16 = 3.375$, $f^* = 2260$

SP122 $(n_1 = 3, n_2 \geqslant 3)$ has no feasible solutions.

Now we branch subproblem SP121 into two new subproblems, one with constraint $n_2 = 3$ and one with constraint $n_2 \geqslant 4$. This leads to:

SP1211 $(n_1 \leqslant 2, n_2 = 3)$ results in $n_1^* = 2$ and $n_2^* = 3$, $f^* = 2080$

SP1212 $(n_1 \leqslant 2, n_2 \geqslant 4)$ has no feasible solutions.

It is clear that now the tree search has been completed and that the global optimum $f^* = 2240$ is reached for $n_1^* = 4$, $n_2^* = 2$. In this two-parameter case the problem can easily be solved. However, if the number of parameters grows, so does the complexity, and the problem becomes unsolvable (as was explained in Section 11.1). $\square$

11.3.2 Dynamic programming

This is a technique that is typically used to solve multistage decision problems. It involves two basic ideas:

1. Bellman's principle of optimality: "in a multistage decision process, given any current state, the remaining sequence of decisions forms an optimal policy with this given state regarded as the initial state. That is to say, whatever the state and decision that led to this state, all future decisions are optimal".

2. The principle of embedding: "instead of attempting to solve a difficult problem directly, one embeds the problem in a family of simpler, easier to solve problems and obtains the solution to the original problem as a result of the solutions to the problems in the family".

The most important requirement when using this method is the ability to define and relate the state of the system to the state of the search process.

The algorithm decomposes the multistage decision problem into a number of subproblems which calculate the optimal path between all states before and after one decision stage.

A classical way of viewing dynamic programming is as a technique that uses problem decomposition. However, alternatively, dynamic programming can also be viewed as a breadth-first search which uses branch merging (see Figure 11.5).

Each stage in the multi-stage decision process is one "generation" of nodes in the search tree. Dynamic programming solves the stages sequentially, which means that:

- It performs a breadth-first search.

- The selection of choices is constrained to be the one of that decision stage. In other words, the search is constrained to be constructive.

Furthermore,

- The principle of optimality is used to prove that different subtrees are isomorphic; and that related nodes can be merged.

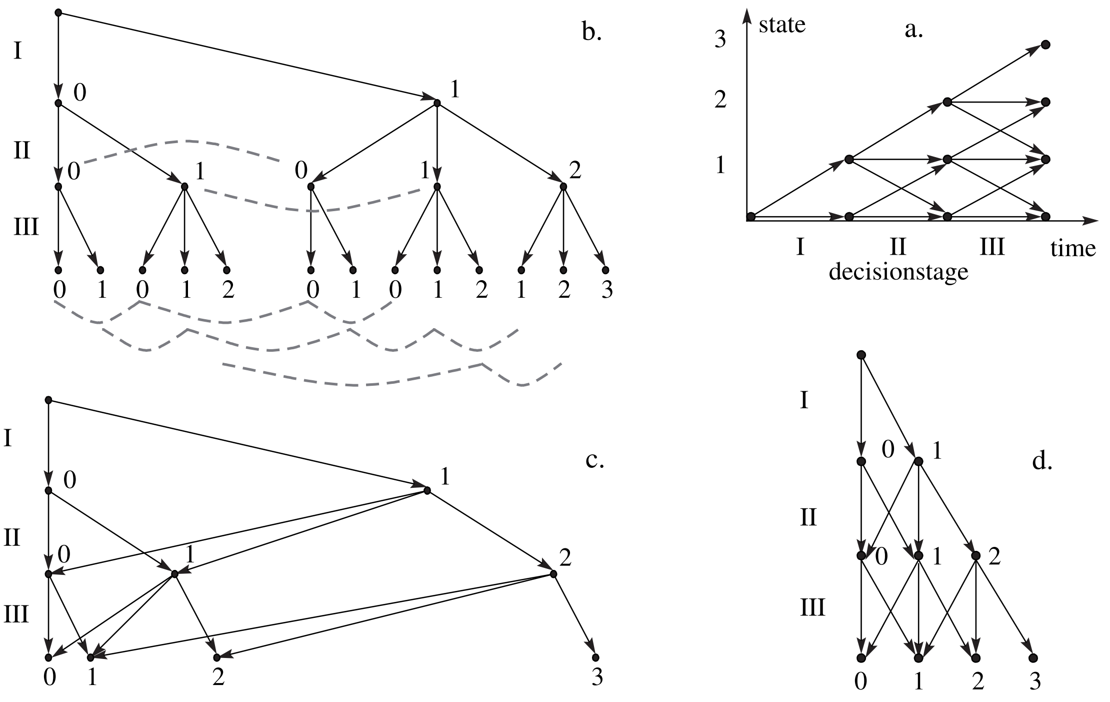


Figure 11.5: Dynamic programming can be viewed as breadth-first search using branch merging. a) The "classical" dynamic programming network in which the stages are calculated serially. b) The same network written as a tree. The concept of "state" in combination with the principle of optimality is used to prove that certain subtrees are isomorphic, to be able to merge their nodes. c) The tree b after branch merging. d) Compactly drawing the tree of c results in d.

11.3.3 Constraint Logic Programming (CLP)

These methods transform the optimization problem into one discrete-domain Constraint Satisfaction Problem (CSP) and use Constraint Logic Programming (CLP) methods to search for a feasible or sometimes optimal solution (using constraint propagation and branch-and-bound search).

Most CLP tools perform a depth-first search with automatic backtracking. The VVO consists of a set of either generic or user-defined heuristics. The most important feature of these tools is the use of constraint propagation which also performs the feasibility test of nodes. Most CLPs do not evaluate optimality and therefore search only for feasible solutions.

Node consistency:

If the domain of a variable contains a value a that does not satisfy one of the constraints, then all leaves in the branches originating from nodes with the value a do not have to be considered any more.

Arc consistency:

The idea of node consistency can be extended to a combination of two variables, and we only consider branches originating from a node with a feasible combination of two variables.

k-**consistency:**

Of course one can consider the feasibility of a combination of k variables, and select only branches which may give feasible solutions. The extension to the full n-consistency in case of a n-dimensional vector would give the whole feasible set in one time. However, the computation time needed to compute the n-consistency set is also exponential and a balance must be made between the degree of consistency maintenance and backtracking.

Example 11.2 Constraint Propagation. Consider the brewery scheduling problem of Chapter 2 and of Example 11.1 earlier on in this section. To see the full effect of constraint propagation we introduce the variable $n_3 = n_1 + n_2$ to denote the total number of pallets of beer produced, and substitute $n_2 = n_3 - n_1$. Now the goal is to maximize profit:

$$\max_{n_1,n_3} \; -160n_1 + 480n_3$$

subject to the constraints

$$\begin{array}{rcrcll} & & 16n_3 & \leqslant & 100 & (C_1) \\ -1.6n_1 & + & 3.2n_3 & \leqslant & 14 & (C_2) \\ & & n_1, n_3 & \geqslant & 0 & (C_3) \\ & & n_3 - n_1 & \geqslant & 0 & (C_4) \\ & & n_1,\, n_3 & \in & \mathbb{Z} & (C_5) \end{array}$$

The domain for the variables is given by

$$n_1 \in \mathbb{Z}$$

$$n_3 \in \mathbb{Z}$$

Now we consider node-consistency, first with constraint C_1 and C_3. We find:

$$\begin{aligned} n_3 &\in \{0,1,2,3,4,5,6\} \\ n_1 &\in \{0,1,2,\dots\} \;. \end{aligned}$$

Next we consider arc-consistency. From the constraints C_2 and C_4 we find the candidate solution set with the 22 elements.

$$\begin{array}{rlllll} (n_1,n_3) \in \{ & (0,0), & (0,1), & (1,1), & & \\ & (0,2), & (1,2), & (2,2), & & \\ & (0,3), & (1,3), & (2,3), & (3,3), & \\ & (0,4), & (1,4), & (2,4), & (3,4), & (4,4), \\ & (2,5), & (3,5), & (4,5), & (5,5), & \\ & (4,6), & (5,6), & (6,6)\} & & \end{array}$$

We can now compute the objective function for all 22 candidate solutions and we find the optimum $f^* = 2240$ for $n_1^* = 4$, $n_3^* = 6$. The method to find the optimum in this search-tree is not considered here because the limited number of candidate solutions. For most integer optimization problems the choice of the VVO is of decisive importance for the speed of the search. □

11.3.4 Heuristic search techniques

A heuristic is defined as a technique which seeks good (i.e., near-optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is. Examples of heuristic search techniques are simulated annealing, genetic algorithms, random search, tabu search, and artificial neural networks.

Heuristic techniques are generally recent developments that originated from the field of computational intelligence. Apart from artificial neural networks, they generate candidate solutions and use a simulation to evaluate the criterion function and the feasibility. The main difference among these types of algorithms is the way in which they generate the new candidates:

- *Random search/Monte Carlo search*: As the name indicates, the selection of candidates from the search space is purely random. This is the most "primitive" method.

- *Simulated annealing* can be viewed as a local hill climber, using a kind of gradient. However, to avoid getting stuck in a local minimum, there is a probability that the "wrong" (i.e., not improving) direction is used. This probability decreases in time. Practice shows that this strategy is more efficient than random search.

- *Tabu search* [17] is similar to simulated annealing, i.e., it is also a hill climber, but in contrast to simulated annealing it is not stochastic but deterministic. It keeps track of a tabu list of "moves" which are not allowed even when they might result in an improved solution. The tabu list contains the previously performed moves. By not selecting them again, returning to a previously found local optimum is prevented.

- *Genetic algorithms* [11, 18]: Instead of one candidate solution, a set (i.e., a generation) of candidate solutions is generated. All members of the set are evaluated. The best are more likely to be used to generate a new set by a crossover with other members and mutation. A genetic algorithm is a type of stochastic search algorithm and in general it performs better than random search since somehow the algorithm gathers information on the search space while searching.

- *Artificial neural networks* principally differ from the methods mentioned above because they do not use direct generation of leaves. Instead, this method performs some kind of local (hill-climbing) unconstrained search. The approach uses a dynamic artificial neural network in which the output of the neurons is fed back to the input. As a consequence, it becomes a dynamic system. Under certain conditions, the dynamic artificial neural network will settle in a stable steady state which is a minimum of an energy function. The trick of using an artificial neural network for optimization consists of two steps:

    1. Transforming the constrained, discrete optimization problem into an unconstrained continuous optimization problem by adding penalty functions (see Section 5.2.3) on constraint violations to the original criterion function.
    2. Defining an artificial neural network such that its energy function represents the (new) criterion function. The problem is, of course, that the energy function will have local minima and, just as in any local search, the one which will be found depends on the initial state of the artificial neural network. The advantage is that the computation can be done in parallel (e.g., one computing node for each neuron) or as an analog circuit.

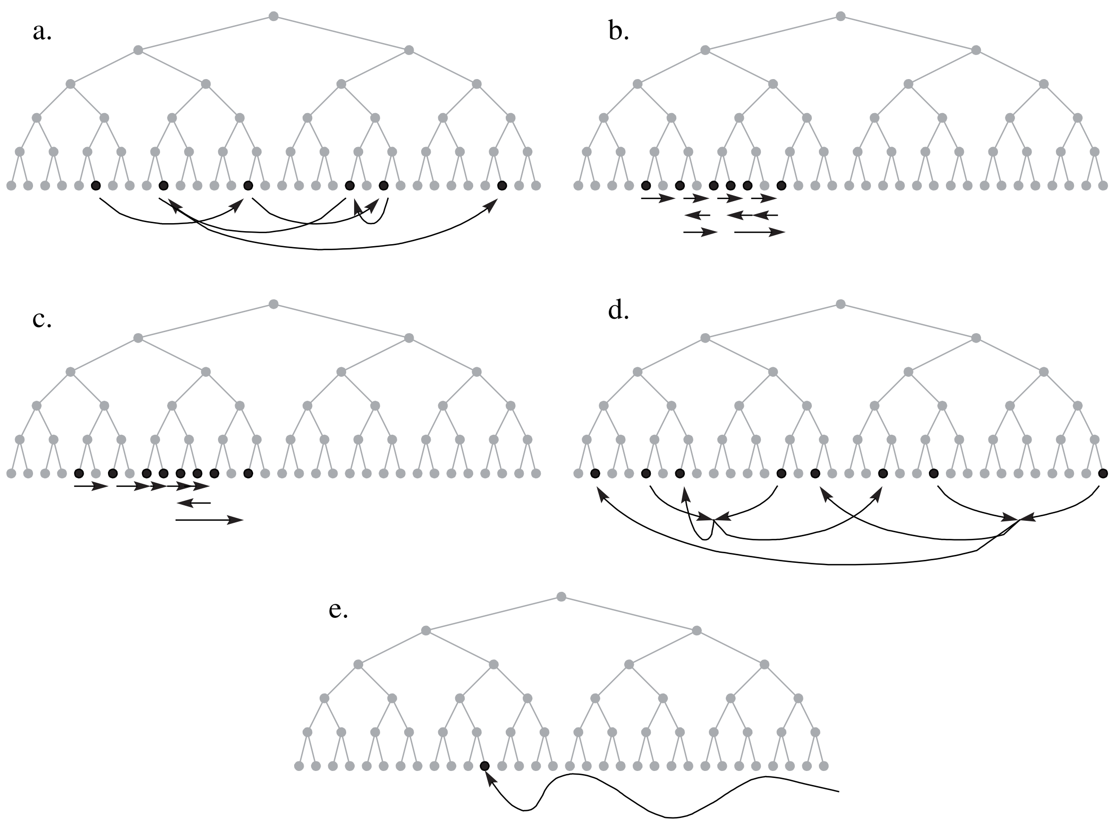


Figure 11.6: Heuristic search: a) Random search. Each next candidate is selected randomly. b) Simulated annealing. Local search with probability of traveling in "wrong" direction. c) Tabu search. Local search with tabu list of recently made moves. d) Genetic algorithm. New generation of candidates are generated by crossover of members of the previous generation. e) Artificial neural networks. Local search in an unconstrained continuous transformation of the original problem.

The list of integer optimization techniques presented in this section makes no pretense to be complete. It gives an idea of the main solution techniques that are currently being used and researched. The approaches represent some typical implementations in their categories. Many modern implementations cannot be categorized under one specific heading. Instead they combine techniques from different categories. This is a very logical development because, as is the main message of this section, these techniques seem to be special cases of a generalized tree search technique.

Part II

Formulating the Controller Design Problem as an Optimization Problem

Chapter 12

Multi-Criteria Controller Design: The LTI SISO Case

In this chapter we introduce the relevant issues in designing a controller via the use of numerical optimization schemes and define the configuration of a single degree of freedom closed-loop feedback system. In Section 12.1 we start with an outline of the relevant issues in the controller design process and by means of an example we illustrate that the optimization problem related to a very simple control design easily becomes highly nonlinear and non-convex. Nonlinear non-convex problems are not attractive for a number of different reasons, one of which is the difficulty to find the global minimum and — related to that — the high numerical complexity to find such a minimum. Much more attractive from a numerical perspective are *convex* optimization problems. This is shown by comparing the computational complexity of *convex* optimization problems with non-convex, nonlinear optimization problems in general. In Section 12.2 we define the configuration of a single degree of freedom closed-loop feedback system. Although the definition is restricted to the single-input, single-output (SISO) case, the recasting of this configuration into a general feedback formulation, done in Section 12.3, is applicable for the multi-input, multi-output (MIMO) case as well. In Section 12.3 we make a first attempt to formalize the multi-criteria feedback controller design problem. The important constraint that the controller stabilizes the feedback configuration is the first constraint that is discussed in more detail. This is done in Section 12.4.

The problem will be formulated in discrete time. Here we use k as the time index and $G(z)$ would denote the z-transform of the impulse response function $g(k)$ of a linear time-invariant (LTI) system.

In the following chapter, we then discuss a number of performance measures (controller design objectives) that lead to a convex optimization problem.

12.1 Introduction

12.1.1 Issues in control system design

The process of designing a control system generally involves many steps. A typical scenario is as follows:

1. Study the system to be controlled and decide what types of sensors and actuators will be used and where they will be placed.

2. Model the resulting system to be controlled.

3. Simplify the model if necessary so that it is tractable.
4. Analyze the resulting model; determine its properties.
5. *Decide on performance specifications.*
6. *Decide on the type of controller to be used.*
7. *Design a controller to meet the specifications (specs), if possible; if not, modify the specs or generalize the type of controller sought.*
8. Simulate the resulting controlled system, either on a computer or in a pilot plant.
9. Repeat starting from step 1 if necessary.
10. Choose hardware and software and implement the controller.
11. Tune the controller on-line if necessary.

In these lecture notes we focus on the items 5–7. More precisely,

ad. 5. How do we express an engineering spec such as: the overshoot of the step response of the controlled system should be smaller than a pre-specified limit, into a mathematical optimization problem?

ad. 6. The relevant issue here is what the consequences are of the choice of a controller structure (e.g., PID), within the class of linear controllers, on the complexity of the related optimization problem.

ad. 7. Almost as important as finding the global optimum of the optimization problem, is the insight that particular specs *cannot* be achieved with the assumed controller/sensor configuration.

It must be kept in mind that a control engineer's role is not merely one of designing control systems for fixed plants, of simply "wrapping a little feedback" around an already fixed physical system. It also involves assisting in the choice and configuration of hardware by taking a system-wide view of performance.

It is also important to realize that practical problems deal with uncertain, non-minimum-phase plants (for linear, discrete-time systems, non-minimum-phase means the existence of zeros outside the unit circle, so the inverse is unstable); that there are inevitable unmodeled dynamics that produce substantial uncertainty, usually at high frequencies; and that sensor noise and input signal level constraints limit the achievable benefits of feedback. A theory that excludes some of these practical issues can still be useful in limited application domains. For example, many process control problems are so dominated by plant uncertainty and non-minimum-phase zeros that sensor noise can be neglected. Some spacecraft problems, on the other hand, are so dominated by trade-offs between sensor noise, disturbance rejection, and input signal level (e.g., fuel consumption) that plant uncertainty and non-minimum-phase effects are negligible. Nevertheless, any general theory should be able to treat all these issues explicitly and give quantitative and qualitative results about their impact on system performance.

In this section we look at two issues involved in the design process: deciding on the performance specifications and modeling. We begin with an example to illustrate these two issues.

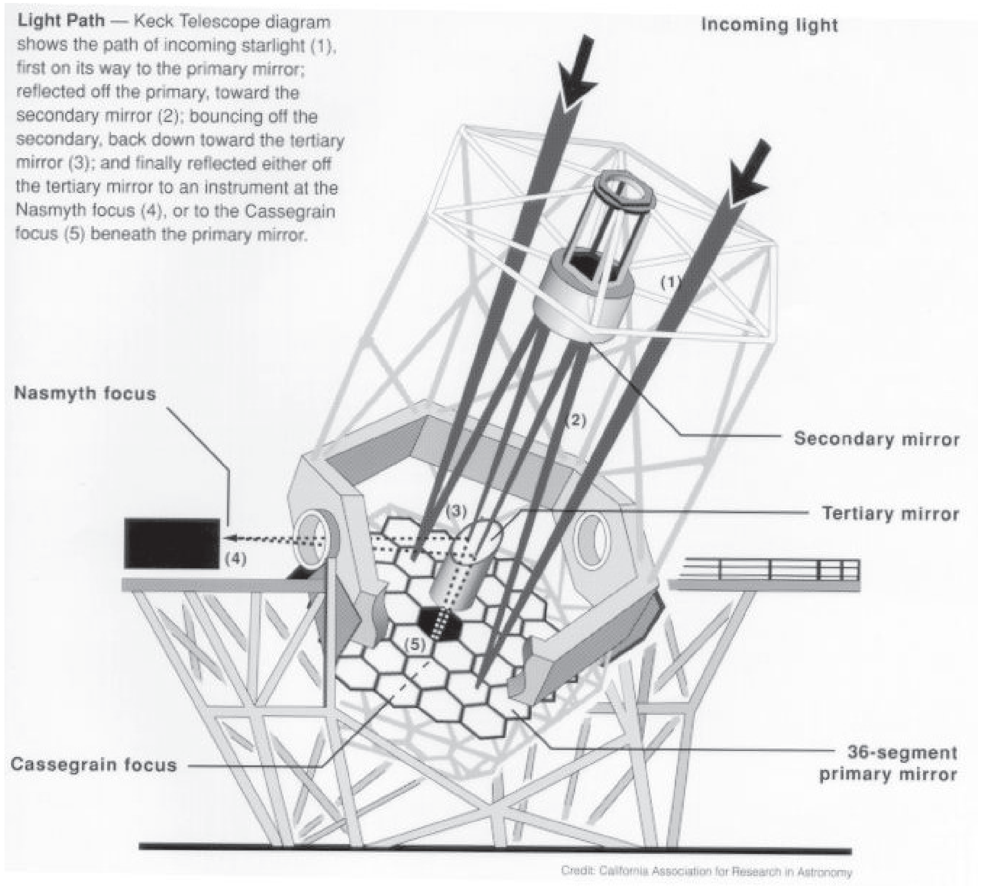

Light Path — Keck Telescope diagram shows the path of incoming starlight (1), first on its way to the primary mirror; reflected off the primary, toward the secondary mirror (2); bouncing off the secondary, back down toward the tertiary mirror (3); and finally reflected either off the tertiary mirror to an instrument at the Nasmyth focus (4), or to the Cassegrain focus (5) beneath the primary mirror.
Incoming light
(1)
(2)
(3)
(4)
(5)
Nasmyth focus
Secondary mirror
Tertiary mirror
Cassegrain focus
36-segment primary mirror
Credit: California Association for Research in Astronomy

Figure 12.1: Keck telescope; one of the largest astronomical telescopes in the world.

Example 12.1 A very interesting engineering system [13] is the Keck astronomical telescope, built on Mauna Kea in Hawaii. It is one of the world's largest telescopes. The basic objective of the telescope is to collect and focus starlight using a large mirror. The shape of the mirror determines the quality of the observed image. The larger the mirror, the more light that can be collected, and hence the dimmer the star that can be observed. The diameter of the mirror on the Keck telescope is 10 m. To make such a large, high-precision mirror out of a single piece of glass would be very difficult and costly. Instead, the mirror on the Keck telescope is a mosaic of 36 hexagonal small mirrors. These 36 segments must then be aligned so that the composite mirror has the desired shape (see Figure 12.1).

The control system to do this is illustrated in Figure 12.2.

As shown, the mirror segments are subject to two types of forces: disturbance forces (described below) and forces from actuators. Behind each mirror segment there are three piston-type actuators, applying forces at three points on the segment to affect its orientation. In controlling the shape of the mirror, it suffices to control the misalignment between adjacent mirror segments. In the gap between every two adjacent segments are (capacitor-type) sensors measuring local displacements between the two segments. These local displacements are stacked into the vector labeled y; this is what is to be controlled. For the mirror to have the ideal shape, these displacements should have certain ideal values that can be pre-computed; these are the components of the vector r. The controller must be designed so that in the closed-loop system y is held close to r despite the disturbance forces. Notice that the

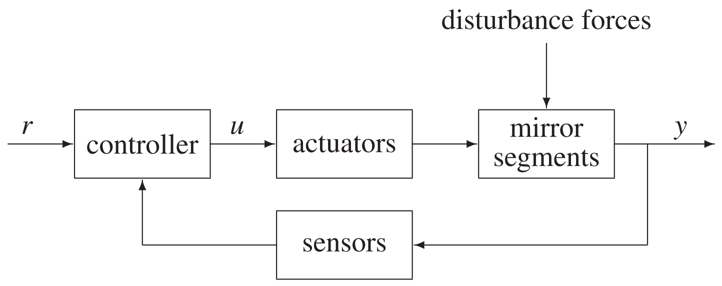


Figure 12.2: Block diagram of Keck telescope control system.

signals are vector valued. Such a system is *multivariable*. Our uncertainty about the plant arises from disturbance sources:

- As the telescope turns to track a star, the direction of the force of gravity on the mirror changes. This introduces nonlinear behavior. Therefore, when employing a linear model, the deviation between linear dynamics and the nonlinear ones acts as a perturbation to the linear model.
- During the night, when astronomical observations are made, the ambient temperature changes. This changes the dynamical properties of the mirror segments.
- The telescope is susceptible to wind gusts.

and from uncertain plant dynamics:

- The dynamic behavior of the components — mirror segments, actuators, sensors — cannot be modeled with infinite precision.

$\square$

Generally speaking, the objective in a control system is to make some output, say y, behave in a desired way by manipulating some input, say u. The simplest objective might be to keep y small (or close to some equilibrium point) — a *regulator problem*— or to keep $y-r$ small for r, a reference or command signal, in some set — a *servomechanism* or *servo problem*. There might be the side constraint of keeping u itself small as well, because it might be constrained (e.g., the flow rate from a valve has a maximum value, determined when the valve is fully open) or it might be too expensive to use a large input. But what is small for a signal? It is natural to introduce norms for signals; then "y small" means "$\|y\|$ small". Which norm is appropriate depends on the particular application.

In summary, performance objectives of a control system naturally lead to the introduction of norms; then the specs are given as norm bounds on certain key signals of interest.

12.1.2 An example of formulating a controller design problem as an optimization problem

Example 12.2 Consider the feedback configuration depicted in Figure 12.3 with

$$G(q) = \frac{27+36q^{-1}+24q^{-2}}{9+6q^{-1}+2q^{-2}} \qquad \text{and} \qquad K(q) = \frac{\kappa}{1+pq^{-1}}$$

The signal e is zero-mean white noise and q^{-1} is the delay operator (so $q^{-1}u(k) = u(k-1)$). The controller objective is to determine $\kappa, p \in \mathbb{R}$ such that:

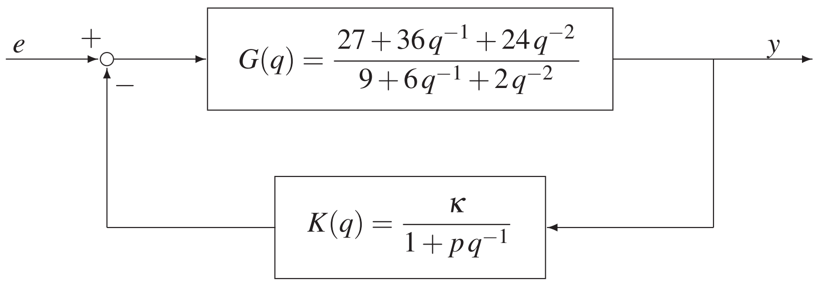

e
$+$
$-$
$G(q) = \frac{27+36q^{-1}+24q^{-2}}{9+6q^{-1}+2q^{-2}}$
y
$K(q) = \frac{\kappa}{1+pq^{-1}}$

Figure 12.3: A controlled second-order linear time-invariant system.

1. The closed-loop configuration is stable.
2. The variance of the controlled output y is minimized.

Both engineering specs will now be transformed into mathematical conditions.

ad. 1. The transfer function of the closed-loop configuration in Figure 12.3 is given by:

$$y(k) = \frac{G(q)}{1+G(q)K(q)}e(k) \tag{12.1}$$

The denominator of this transfer function equals:

$$\kappa(27+36q^{-1}+24q^{-2})+(9+6q^{-1}+2q^{-2})(1+pq^{-1}) \tag{12.2}$$

The condition that the closed-loop system in Figure 12.3 is internally stable (see next subsection) is equivalent to polynomial (12.2) having all its roots within the unit circle. This condition can be checked using Jury's rule (equivalent to the continuous-time Routh test [1]; see also Appendix B). In general it is a difficult problem to determine whether a system of polynomial equations in several variables such as this can be satisfied. For our particular case the region of parameters which give closed-loop stability is shown in Figure 12.4.

The two disconnected regions clearly show non-convexity. This figure was produced by determining the roots of the closed-loop transfer function for different values of p, κ and checking stability of these roots. This extremely specific problem, with only two free parameters, required considerable effort to solve. The case in which the controller K would depend on more than two parameters would be extremely difficult to solve.

ad. 2. For this simple example, it is possible to express the variance of the output of the feedback system in Figure 12.3 in analytic terms using basic concepts from signal analysis [27]. Let the closed-loop configuration be given by the state-space realization

$$\begin{aligned} x(k+1) &= A_{\text{cl}}x(k)+B_{\text{cl}}e(k) \\ y(k) &= C_{\text{cl}}x(k)+D_{\text{cl}}e(k) \end{aligned}$$

where A_{cl}, B_{cl}, C_{cl} and D_{cl} depend on the parameters κ and p. Further let σ_e^2 denote the variance of the white noise input sequence, then the *auto-covariance function* is given by [1]

$$\text{RMS}^2(y) = E[y(k)y^T(k)] = C_{\text{cl}}XC_{\text{cl}}^T + D_{\text{cl}}D_{\text{cl}}^T\sigma_e^2$$

[1]The equations will be derived in Section 14.6.

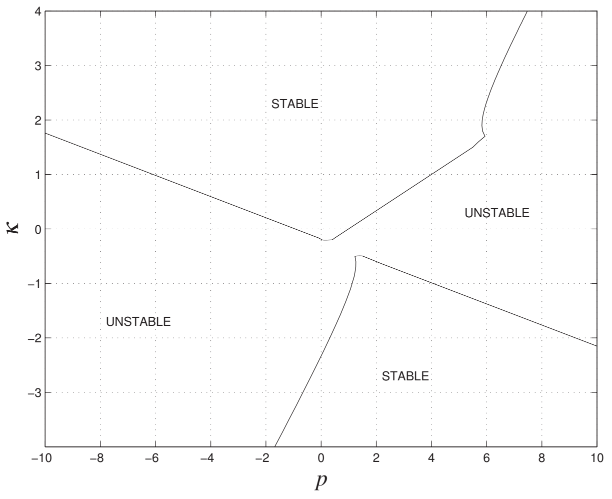


Figure 12.4: The region of controller parameters κ, p that give closed-loop stability for the configuration in Figure 12.3.

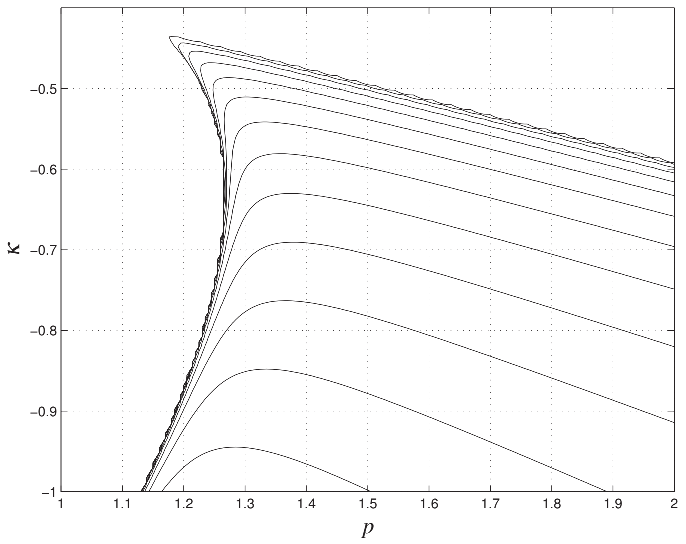


Figure 12.5: Contour lines of RMS(y) as a function of the parameters κ and p.

where X is the solution of the discrete-time Lyapunov equation

$$X = A_{\mathrm{cl}} X A_{\mathrm{cl}}^T + B_{\mathrm{cl}} B_{\mathrm{cl}}^T \sigma_e^2$$

In Figure 12.5, we depict the contour plot of RMS(y) for $\kappa \in [-1, -0.4]$ and $p \in [1, 2]$. This function is clearly not convex.

□

The example shows that even in the linear, time-invariant case imposing constraints on the controller configuration can lead to highly complex, nonlinear and non-convex optimization problems. The drawback of such types of optimization problems is highlighted briefly in the next section.

12.1.3 Importance of convexity in optimization problems

By far the most important attribute of convex optimization problems is that there are effective methods for solving them [4].

This assertion can be argued on several levels. On a pragmatic (every day practice, not theory) level, we note that very large linear, (convex) quadratic, and convex programs are routinely solved numerically; the growth of computation time with the numbers of variables and constraints has been observed empirically to be quite moderate (much less than combinatorially). In contrast, *global* solutions of non-convex programs are attempted only occasionally, and in general require extremely large computation times, which in addition tend to rise combinatorially with the number of variables [35].

For this reason, numerical optimization of general non-convex programs is mostly restricted to the computation of *local* optima, or heuristic methods to compute the global optimum. In many cases, these local or heuristic methods produce acceptable solutions, which may even be globally optimal, but this cannot be guaranteed.

It is also possible to put this notion that convex optimization problems are "substantially more tractable" than non-convex problems on a firm theoretical ground. We very roughly paraphrase some of the results from the book by Nemirovsky and Yudin [31], to which we refer the reader for complete, precise statements. The minimum number of computations (function and gradient or subgradient evaluations) required to compute the *global* minimum of a general differentiable function in n variables, within an accuracy of ε, necessarily grows like,

$$\left(\frac{1}{\varepsilon}\right)^n$$

i.e., roughly speaking, the complexity of a search over an ε-grid.

If the function is convex, however, the minimum number of "computations" necessary to compute an ε-approximation of the global minimum has more the flavor of

$$n \log\left(\frac{1}{\varepsilon}\right)$$

i.e., roughly speaking, the complexity associated with a bisection method. The reader should note the slow growth of this number with both accuracy ε and the number of variables n. Taking, e.g., ε equal to 10^{-3} and $n = 10$, we have the following complexity measures:

$$\text{For obtaining the global minimum of a non-convex function}: \quad 10^{30}$$

$$\text{For obtaining the global minimum of a convex function}: \quad 30$$

12.2 The basic feedback loop

The most elementary feedback control system has three components: a plant (the object to be controlled, no matter what it is, is always called the *plant*), a sensor to measure the output of the plant, and a controller to generate the plant's input. Usually, the actuators are integrated in the plant and are considered part of it. A block diagram of such an elementary feedback control system is given in Figure 12.6. Here we notice that each block has two inputs, one internal to the system and one coming from outside, and one output. These signals have the following interpretations:

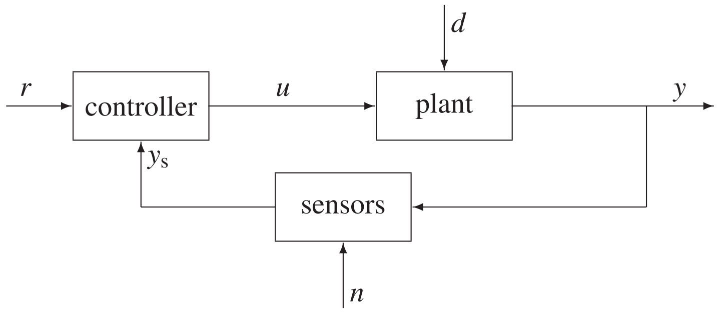


Figure 12.6: *Schematic representation of an elementary feedback system.*

r	command input
y_s	sensor output
u	plant input
d	external disturbance
y	plant output
n	sensor noise

The three signals coming from outside, i.e., r, d and n, are called the *exogenous inputs*.

The performance specifications are generally given in the following terms: the plant's output y should approximate some pre-specified function of r, and it should do so in the presence of the disturbance d, sensor noise n, etc. We may also want to limit the size of u. Frequently, it makes more sense to describe the performance objective in terms of the measurement y_s rather than y, since often the only knowledge of y is obtained from y_s.

The equations that describe the feedback configuration in Figure 12.6 can be easily obtained in the frequency domain. Let the plant's equation be denoted as follows,

$$y(k) = P(q)u(k) + P(q)d(k)$$

where q denotes the forward-shift operator (so $q\,u(k) = u(k+1)$). We shall take an even more specialized viewpoint and suppose that the outputs of the three components are linear functions of the sums (or difference) of their inputs and further assume the sensor transfer to be equal to one ($F(q) = 1$ for all q); that is, the plant, sensor, and controller equations are taken to be of the form,

$$\begin{aligned} y(k) &= P(q)\Big(d(k) + u(k)\Big) \\ y_s(k) &= F(q)\Big(y(k) + n(k)\Big) \\ u(k) &= K(q)\Big(r(k) - y_s(k)\Big) \end{aligned}$$

The minus sign in the last equation is a matter of tradition. The block diagram for these equations is given in Figure 12.7. The feedback loop in Figure 12.7 is physically feasible when it is in mathematical terms called *well-posed*. This means that all closed-loop transfer functions exist, that is, all transfer functions from the three exogenous inputs (r, d, n) to all internal signals (x_1, x_2, x_3) exist. For $F = 1$

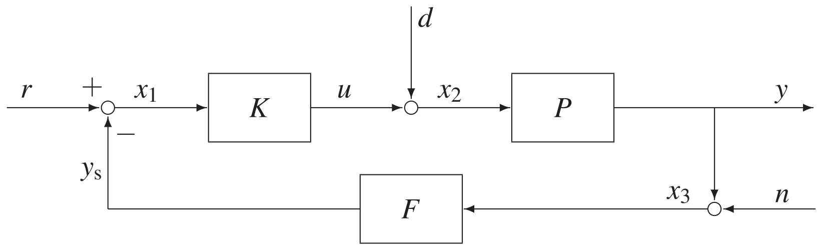


Figure 12.7: *Basic feedback loop. The internal signals x_1, x_2 and x_3 are indicated. Note the negative feedback (indicated by the minus sign near the arrow that feeds the y_s signal to the summator on the left-hand side of the block diagram). Also note the convention of leaving out the plus signs near the arrows feeding into the summator that constructs x_2 as $x_2 = u + d$.*

we can write the following equations:

$$\begin{aligned} x_1(k) &= r(k) - x_3(k) \\ x_2(k) &= d(k) + K(q)x_1(k) \\ x_3(k) &= n(k) + P(q)x_2(k) \end{aligned}$$

or in matrix form,

$$\begin{bmatrix} 1 & 0 & 1 \\ -K & 1 & 0 \\ 0 & -P & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} r \\ d \\ n \end{bmatrix}$$

Thus the system is well-posed if and only if the above 3×3 matrix is nonsingular, that is, the determinant, $1 + PK$, is not identically zero. For example, the system with $P(q) = 1, K(q) = -1$ is not well-posed. Then the nine transfer functions are obtained from the equation,

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \frac{1}{1+PK} \begin{bmatrix} 1 & -P & -1 \\ K & 1 & -K \\ PK & P & 1 \end{bmatrix} \begin{bmatrix} r \\ d \\ n \end{bmatrix} \tag{12.3}$$

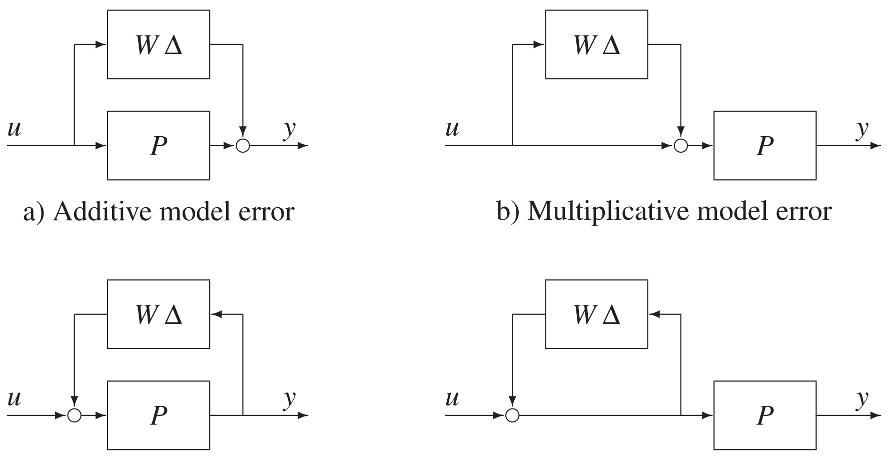
A stronger notion of well-posedness that makes sense when P and K are proper is that the nine transfer functions above are proper[2]. A necessary and sufficient condition for this is that $1 + PK$ is proper, but not strictly proper.

12.2.1 Plant uncertainty

Doyle, Francis and Tannenbaum state in their book on feedback control theory [13]:

> *No mathematical system can exactly model a physical system. For this reason we must be aware of how modeling errors might adversely affect the performance of a control system.*

[2] A transfer function $H(z)$ is called proper if $\lim_{z\to\infty} H(z)$ is finite and strictly proper if $\lim_{z\to\infty} H(z) = 0$.



a) Additive model error b) Multiplicative model error

c) Reverse additive model error d) Reverse multiplicative model error

Figure 12.8: Model error structures

If there are modeling errors, the model P does no longer accurately describe the system to be controlled. We can define around the nominal model P a set $\mathscr{P}$ of *perturbed plants*. This set then takes into account the modeling errors. An important question coming up is: when does the controller K also stabilize the perturbed plants in the set $\mathscr{P}$?

In order to make some precise statements about robust stability, we have to parametrize the set $\mathscr{P}$ of perturbed plants. In modeling we arrive at a certain set of perturbed plants. This set may not be easy to describe mathematically, so we may embed it in a larger set that is easier to handle. In this section we will consider some model error configurations to describe perturbed plants. Consider a perturbed plant $\tilde{P}$ and assume we have the nominal model P available.

A very common uncertainty description is the **additive model error structure** in which the uncertainty Δ is assumed to be additive to the nominal model P:

$$\tilde{P}(q) = P(q) + W(q)\Delta(q)$$

where W is a known weighting filter such that the uncertainty Δ becomes bounded as[3]

$$\| \Delta \|_\infty = \sup_\omega |\Delta(e^{j\omega})| < 1 \quad .$$

Another common uncertainty description is the **multiplicative model error structure** in which the uncertainty Δ is taken to be multiplicative to the nominal model P:

$$\tilde{P}(q) = P(q)\Big(1 + W(q)\Delta(q)\Big)$$

where W is a known weighting filter such that the uncertainty becomes bounded as

$$\| \Delta \|_\infty < 1 \quad .$$

[3]In Section 13.4 we will formally define the ∞-norm of a transfer function.

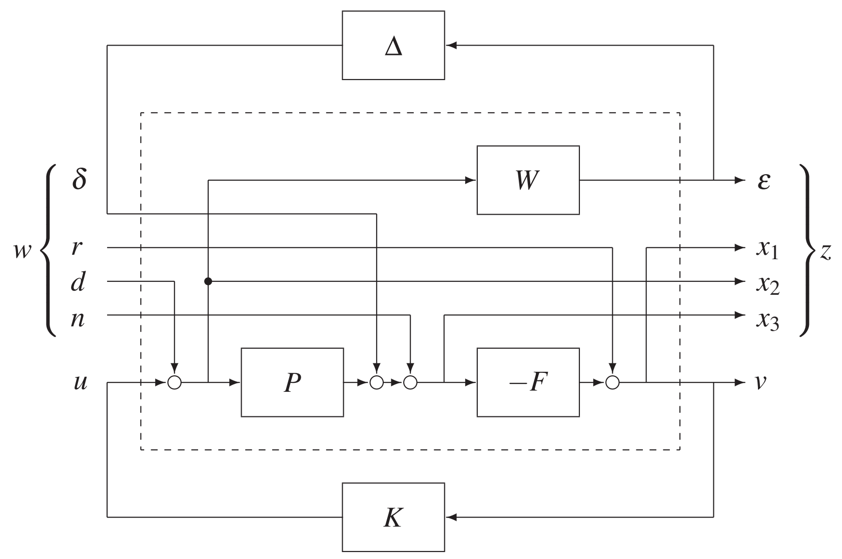


Figure 12.9: Uniform representation of a feedback loop.

Other possibilities are the **reverse additive model error structure** and the **reverse multiplicative model error structure** in which the uncertainty Δ is in a feedback loop

$$\begin{aligned}\text{reverse additive:}\quad \tilde{P}(q) &= P(q)\Big(1+W(q)\,\Delta(q)P(q)\Big)^{-1}\\ \text{reverse multiplicative:}\quad \tilde{P}(q) &= P(q)\Big(1+W(q)\,\Delta(q)\Big)^{-1}\end{aligned}$$

where W is a known weighting filter and the uncertainty is bounded as $\|\Delta\|_\infty < 1$.

The configurations of the four types of model error structures are given in Figure 12.8.

12.3 General formulation of the basic feedback loop

Next, we give a general formulation for the basic feedback loop represented in Figure 12.7, such that it represents a wide class of controller design problems (even for MIMO systems). For this purpose we redraw Figure 12.7 into Figure 12.9, where we also take an additive model error structure into account. If we now define the following vector quantities,

$$w(k) = \begin{bmatrix} \delta(k) \\ r(k) \\ d(k) \\ n(k) \end{bmatrix} \qquad z(k) = \begin{bmatrix} \varepsilon(k) \\ x_1(k) \\ x_2(k) \\ x_3(k) \end{bmatrix} \qquad v(k) = x_1(k) \;, \tag{12.4}$$

then Figure 12.9 can be represented as in Figure 12.10 as the “most general control system”, so

$$\begin{bmatrix} z(k) \\ v(k) \end{bmatrix} = G(q) \begin{bmatrix} w(k) \\ u(k) \end{bmatrix} \tag{12.5}$$

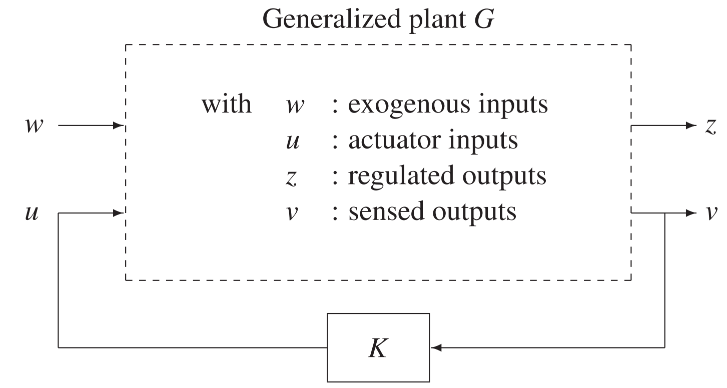


Figure 12.10: Generalized plant with controller.

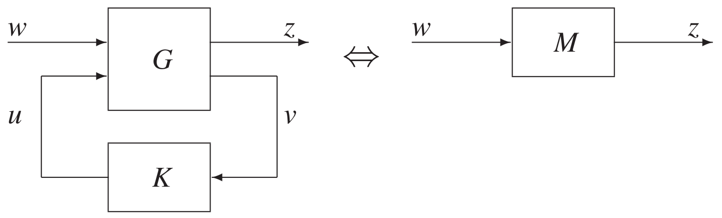


Figure 12.11: General feedback system

or

$$z(k) = G_{(1,1)}(q)w(k) + G_{(1,2)}(q)u(k) \tag{12.6}$$

$$v(k) = G_{(2,1)}(q)w(k) + G_{(2,2)}(q)u(k) \tag{12.7}$$

Often we omit the uncertainty block $\Delta(q)$ in the scheme (we will deal with Δ in Section 13.5) and we obtain the configuration of Figure 12.10.

The generality stems from the fact that the generalized plant in Figure 12.10 does *not only* contain the actual plant, but also the signals w, z which may include all kinds of perturbations and reference signals. Depending on the problem one wants to address w, z consists of a subset or a variety of the quantities defined above.

The closed loop of generalized plant G and controller K is denoted by M (see also Figure 12.11), so

$$z(k) = M(q)\,w(k) = \left[G_{(1,1)}(q) + G_{(1,2)}(q)\,K(q)\Big(I - G_{(2,2)}(q)K(q)\Big)^{-1} G_{(2,1)}(q)\right] w(k) \ . \tag{12.8}$$

This most general formulation is advantageous for at least two reasons:

1. It allows a uniform treatment of a wide class of specific controller design problems, in particular those that only focus on entries of the big transfer matrix M (which will be determined in Exercise 14.1). For example when we consider disturbance rejection, we focus on the effect (transfer) from the disturbance d or n on the output y and require this transfer to be “minimal” in some sense, to be defined later. When tracking of a reference signal is important, we focus on the transfer from r to y and require this transfer to be “as close as possible” to identity!

2. It allows to assess in general terms the class of controllers that assure that the feedback system is *internally stable* (to be defined in the next section). When we only focus on one single transfer function in the closed-loop configuration we can end up with an unstable design even if the controller stabilizes this single transfer function! An integral treatment as pursued with this most general control system overcomes this deficiency.

As we will outline explicitly in Chapter 13, where we will introduce norms of signals and transfer functions, measuring transfer functions is done by using norms. Therefore, expressing that the transfer (function) from one exogenous input to an internal signal in the most general closed-loop system should be smaller than some pre-specified bound γ_1 in some norm is denoted by

$$\|M_{(i,j)}\| \leqslant \gamma_1$$

where $M_{(i,j)}$ indicates one of the submatrices of the transfer matrix M. If we also want, e.g., the variance of the output y to be minimized in the stationary case and if we want our controller to stabilize the feedback system, a first crude formulation of the multi-criteria controller design problem can be stated as follows:

A multi-criteria controller design problem: *Determine a controller C belonging to the class of internally stabilizing controllers, denoted by $\mathscr{S}$, under the constraints,*

$$\begin{array}{rcl} E[y^2(k)] + \rho E[u^2(k)] & \leqslant & \gamma_1 \\ \|M_{(1,1)}\| & \leqslant & \gamma_2 \\ & \vdots & \end{array}$$

where $E[\cdot]$ denotes statistical expected value.

Three frequently occurring attributes in multi-criteria optimization problems, namely internally stabilizing controllers, simple engineering specifications such as overshoot and reference signal tracking, and norm specifications, will be put in a firm mathematical framework in these notes. In the next section we start with the notion of an internally stabilizing controller.

12.4 Internally stabilizing controllers

In the initial phase of a controller design one generally allows for a more general configuration that leads to a more amenable optimization problem. One such constraint is discussed next.

12.4.1 Definition of internal stability

Consider a system with input u, output y, transfer function $H(q)$ and impulse response $h(k)$. In the time domain the equation is

$$y(k) = \sum_{n=-\infty}^{\infty} h(n-k)u(k)$$

If $|u(k)| \leqslant c$ for all k, then

$$|y(k)| \leqslant \left(\sum_{n=-\infty}^{\infty} |h(n)| \right) c$$

where use is made of

$$\left| \sum_{n=-\infty}^{\infty} f(n) \right| \leqslant \sum_{n=-\infty}^{\infty} |f(n)|$$

This relationship states that the system H is bounded-input, bounded-output stable, if $\sum_{k=-\infty}^{\infty} |h(k)|$, that is the 1-norm of $h(k)$ (see Chapter 13), is bounded.

Definition 12.3 (Internal Stability)
A feedback system is said to be internally stable if all transfer functions from any exogenous input to any internal signal are stable.

So if the nine transfer functions in (12.3) are stable, then the feedback system of Figure 12.7 is internally stable. Note that the stability of the transfer functions from r, d, n to x_1, x_2, x_3 also implies the stability of the transfer functions from r, d, n to the signals u, v and y. A consequence of this definition is that if the exogenous inputs are bounded in magnitude, so are x_1, x_2 and x_3, and hence u, y and v. So internal stability guarantees bounded internal signals for all bounded exogenous signals.

The idea behind the definition of internal stability is that it is not enough to look only at one single input-output transfer function, such as from r to y. This transfer function could be stable, so that y is bounded when r is, and an internal signal could be unbounded, probably causing internal damage to the physical system. This is shown in the following example.

Example 12.4 Take P and K in (12.3) equal to $K(q) = -2\frac{1-2q^{-1}}{1+0.5q^{-1}}$ and $P(q) = \frac{1}{1-2q^{-1}}$, then the transfer from r to y is stable, but that from d to y is not (Check this!). The feedback is therefore not internally stable. □

12.4.2 Parametrization of internally stabilizing controllers when the plant is stable

In this section we study the general feedback system with the block diagram shown in Figure 12.11. Here it is assumed that G is stable and strictly proper, and that K is proper. We will consider stability for the nominal case, so we will assume $\Delta = 0$ in this section.

In the generic multi-criteria optimization problem formulated at the end of Section 12.3, one of the constraints was that the controller (internally) stabilizes the feedback system. A parametrization of such controllers, which also turns out to be convex, is given in Theorem 12.5. We use the symbol $\mathscr{S}$ to represent the family of all stable, proper, real, rational functions (transfer functions). Notice that $\mathscr{S}$ is closed under addition and multiplication: If $F, H \in \mathscr{S}$, then $F + H, FH \in \mathscr{S}$. Also $1 \in \mathscr{S}$. (Thus $\mathscr{S}$ is a commutative ring with identity).

Theorem 12.5 (Family of stabilizing controllers for a stable plant)
Assume that $G \in \mathscr{S}$ and let $\Delta = 0$. The set of all controllers for which the feedback system is internally stable equals

$$\left\{ Q(I + G_{(2,2)}Q)^{-1} : Q \in \mathscr{S} \right\}$$

Proof : ($\subset$) Suppose K achieves internal stability. Let Q denote the transfer function from v to u, that is,

$$Q = K(I - G_{(2,2)}K)^{-1}$$

Then $Q \in \mathscr{S}$ and

$$K = Q(I + G_{(2,2)}Q)^{-1}$$

($\supset$) Conversely, suppose $Q \in \mathscr{S}$ and define

$$K = Q(I + G_{(2,2)}Q)^{-1} \tag{12.9}$$

According to Definition 12.3, the feedback system is internally stable if all the transfer functions from the exogenous input w to the signals z, v, and u are all stable and proper. We derive:

$$\begin{aligned} z(k) &= G_{(1,1)}(q)w(k) + G_{(1,2)}(q)u(k) && (12.10) \\ v(k) &= G_{(2,1)}(q)w(k) + G_{(2,2)}(q)u(k) && (12.11) \end{aligned}$$

Using the fact that

$$u(k) = Q(q)(I + G_{(2,2)}(q)Q(q))^{-1}v(k)$$

we find that

$$\begin{bmatrix} z(k) \\ v(k) \\ u(k) \end{bmatrix} = \begin{bmatrix} G_{(1,1)}(q) + G_{(1,2)}(q)Q(q)G_{(2,1)}(q) \\ G_{(2,1)}(q) + G_{(2,2)}(q)Q(q)G_{(2,1)}(q) \\ Q(q)G_{(2,1)}(q) \end{bmatrix} \begin{bmatrix} w(k) \end{bmatrix}$$

Clearly, with G and Q being stable, this transfer function belongs to $\mathscr{S}$. □

Example 12.6 In this example we study the unity-feedback system with the block diagram shown in Figure 12.7 for $F = 1$. Here it is assumed that P is strictly proper and K is proper. The internally stabilizing controller K is now given by

$$K = \frac{Q}{1 - PQ}$$

According to Definition 12.3, the feedback system is internally stable if and only if the nine transfer functions,

$$\frac{1}{1+PK} \begin{bmatrix} 1 & -P & -1 \\ K & 1 & -K \\ PK & P & 1 \end{bmatrix}$$

relating the input signals r, d, and n to the internal signals x_1, x_2, and x_3, are all stable and proper. After substitution of (12.9) and clearing of fractions, this matrix becomes

$$\begin{bmatrix} 1 - PQ & -P(1-PQ) & -(1-PQ) \\ Q & 1 - PQ & -Q \\ PQ & P(1-PQ) & 1 - PQ \end{bmatrix}.$$

Clearly, these nine entries belong to $\mathscr{S}$. □

In the next chapter (see Theorem 13.10 and Remark 13.11) we show that this parametrization makes that the requirement that the controller should robustly stabilize the closed-loop system will be a *convex* specification!

Note that all nine transfer functions above are affine functions of the free parameter Q; that is, they are of the form $H_1 + H_2QH_3$, for H_1, H_2, H_3 in $\mathscr{S}$.

Let us look at a simple optimization feedback controller design problem.

Example 12.7 Suppose that we want to find a K such that the feedback system (see Figure 12.7) is internally stable and y asymptotically tracks a step r, when $d = 0$, $n = 0$ and $F = 1$. Parametrize K as in Example 12.6. Then y asymptotically tracks a step if and only if the transfer function from r to x_1 (i.e., $1 - PQ$) has a zero at $z = 1$, that is

$$P(1)Q(1) = 1 \ . \tag{12.12}$$

To see this, note that the final value property of the z-transform yields

$$\lim_{k\to\infty}\Big(r(k) - y(k)\Big) = 0 \ \Leftrightarrow \ \lim_{z\to 1}\frac{z-1}{z}\Big(R(z) - Y(z)\Big) = 0$$

and that

$$\lim_{z\to 1}\frac{z-1}{z}\Big(R(z) - Y(z)\Big) = \lim_{z\to 1}\frac{z-1}{z}M_{(1,1)}(z)R(z) = \lim_{z\to 1}M_{(1,1)}(z)$$

because $R(z) = \frac{z}{z-1}$.

Equation (12.12) has a solution Q in $\mathscr{S}$ if and only if $P(1) \neq 0$. Conclusion: The problem has a solution if and only if $P(1) \neq 0$. The family of controllers that achieve this goal can be parametrized as follows:

$$\left\{K = \frac{Q}{1 - PQ} : Q \in \mathscr{S}, Q(1) = \frac{1}{P(1)}\right\}$$

Observe that Q inverts P at DC. Also, you can check that a controller of this form has a pole at $z = 1$.
□

Chapter 13

Convex Controller Design Specifications

13.1 Introduction

Let $\mathscr{M}$ denote the set of all $n_z \times n_w$ transfer matrices, where the number of exogenous inputs n_w and the number of regulated outputs n_z are fixed by the choice of controller configuration. With each design specification $\mathscr{D}_\ell$ we will associate the set $\mathscr{M}_\ell \subseteq \mathscr{M}$ of all $n_z \times n_w$ transfer matrices that meet the design specifications $\mathscr{D}_\ell$. An example of formulating a design specification in this way is now briefly considered. Let $\mathscr{D}_\ell$ denote the design specification that the (i,j)-entry of the closed-loop transfer matrix M has unit DC gain. The corresponding set of acceptable transfer matrices is

$$\mathscr{M}_{\text{DC}} = \left\{ M \in \mathscr{M} : M_{(i,j)}(1) = 1 \right\} \quad .$$

A multi-criteria controller design formulation then corresponds to taking the *intersection* of the associated sets of acceptable closed-loop transfer matrices corresponding to each design specification.

The sets related to a broad class of design specifications for LTI systems have a simple geometric form: convex. Using Definition 1.1 of convex sets, we state the definition of a closed-loop convex design specification.

Definition 13.1 (Closed-loop convex design spec) *A design specification on a general closed-loop configuration is closed-loop convex if the set of transfer matrices that satisfy the design specification is convex.*

In this chapter we will discuss convex design specifications. As an example we will start with an engineering specification with respect to overshoot and an engineering specification with respect to tracking a reference signal. In Section 13.4 we will generalize this to specifications in terms of a norm of a transfer function.

13.2 Engineering specification with respect to overshoot

For example, consider the specification that the overshoot of the step response from w_j to z_i does not exceed α:

$$\mathscr{M}_{\text{os}} = \{ M \in \mathscr{M} : m_{(i,j)}(k) \leqslant \alpha \text{ for all } k \geqslant 0 \} \tag{13.1}$$

where $m_{(i,j)}(k)$ is the inverse z-transform of $\left(M_{(i,j)}(z) \frac{z}{z-1} \right)$.

The set $\mathscr{M}_{\text{os}}$ might be called "hard to describe" in frequency domain terms: there is no simple description of $\mathscr{M}_{\text{os}}$ in terms of pole and zero locations of the transfer function $M_{(i,j)}$ (although there are many approximate descriptions and rules-of-thumb). The overshoot specification seems more naturally expressed in "time domain" terms.

Nevertheless, we can easily verify that $\mathscr{M}_{\text{os}}$ is convex (and thus overshoot specifications are closed-loop convex), even if it is hard to describe. We rewrite (13.1) more explicitly as

$$\mathscr{M}_{\text{os}} = \left\{ M \in \mathscr{M} : \frac{1}{2\pi}\int_{-\pi}^{\pi} \frac{M_{(i,j)}(e^{j\omega})}{e^{j\omega}-1} e^{j\omega k} d\omega \leqslant \alpha \text{ for all } k \geqslant 0 \right\}$$

Suppose that M and $\overline{M}$ are transfer matrices in $\mathscr{M}_{\text{os}}$, and let $M_\lambda = (1-\lambda)M + \lambda\overline{M}$, where $\lambda \in [0,1]$. Then for all $k \geqslant 0$,

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} \frac{(M_\lambda)_{(i,j)}(e^{j\omega})}{e^{j\omega}-1} e^{j\omega k} d\omega =$$

$$(1-\lambda)\underbrace{\frac{1}{2\pi}\int_{-\pi}^{\pi} \frac{M_{(i,j)}(e^{j\omega})}{e^{j\omega}-1} e^{j\omega k} d\omega}_{x_1} + \lambda \underbrace{\frac{1}{2\pi}\int_{-\pi}^{\pi} \frac{\overline{M}_{(i,j)}(e^{j\omega})}{e^{j\omega}-1} e^{j\omega k} d\omega}_{x_2}$$

Hence, since $x_1, x_2 \in \mathbb{R}$ with $|x_1| \leqslant \alpha$ and $|x_2| \leqslant \alpha$, we have

$$(1-\lambda)x_1 + \lambda x_2 \leqslant \alpha$$

and therefore $M_\lambda \in \mathscr{M}_{\text{os}}$.

More generally, and less explicitly, since the z-transform and its inverse are linear operations, convex and affine sets are preserved between the frequency and time domains.

13.3 Engineering specification with respect to tracking a reference signal

Often the primary purpose of a feedback control system is to ensure that one of the regulated outputs, say z_1 is nearly equal to a reference or command signal, say w_1. This is a tracking performance specification. In many cases the reference input will be constant for long periods of time, and occasionally change quickly to a new value or "set point". Since the closed-loop system is LTI, its response to such a change in set-point is readily determined from its response when the reference quickly changes from zero to one at $k = 0$ — a unit step response.

Let $s_{(i,j)}(k)$ denote the unit step response from w_j to z_i. Let us list some typical qualitative specifications on this step response. Immediately following the step, we are concerned with preventing severe overshoot or undershoot (large positive and negative excursions of $s_{(i,j)}(k)$ for small k). We would like a short settling time, that is, we would like the value of $s_{(i,j)}(k)$ to go quickly to steady state and stay within, say, five percent of one. Finally, we would like $s_{(i,j)}(k)$ to exactly settle to one (asymptotic tracking of a step input). As shown in Figure 13.1, all of these specs can be expressed together as follows:

$$s_{\min}(k) \leqslant s_{(i,j)}(k) \leqslant s_{\max}(k) \quad \text{for } k \geqslant 0 \ .$$

Let us verify that this constraint is again closed-loop convex. Consider the set of closed-loop transfer matrices which meet this constraint:

$$\mathscr{M}_{\text{track}} = \{M \in \mathscr{M} : s_{\min}(k) \leqslant s_{(i,j)}(k) \leqslant s_{\max}(k) \text{ for all } k \geqslant 0\} \ .$$

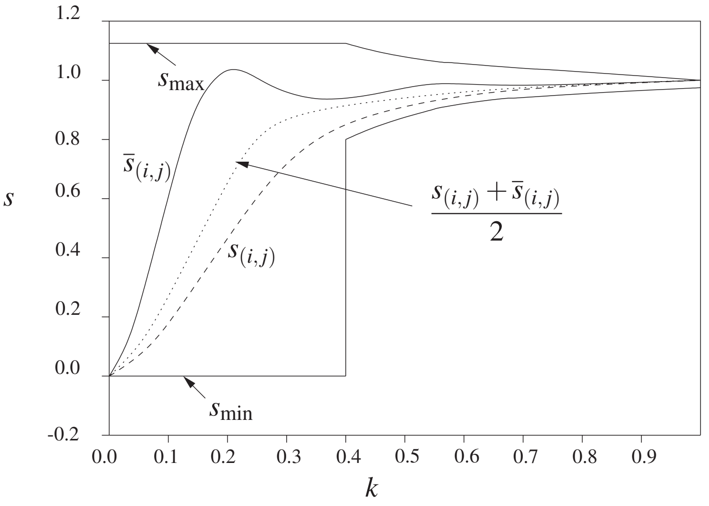


Figure 13.1: Step responses with upper and lower bounds.

Suppose again that M and $\overline{M}$ are transfer matrices in $\mathscr{M}_{\text{track}}$, so that the corresponding step responses $s_{(i,j)}(k)$ and $\bar{s}_{(i,j)}(k)$ lie between $s_{\min}(k)$ and $s_{\max}(k)$ for all k as shown in Figure 13.1. Now if $\lambda \in [0,1]$, then $(s_\lambda)_{(i,j)}(k) = (1-\lambda)s_{(i,j)}(k) + \lambda\bar{s}_{(i,j)}(k)$ also lies in between the specified boundaries. Therefore, $M_\lambda \in \mathscr{M}_{\text{track}}$.

13.4 Engineering specifications in terms of norms of transfer functions

One way to describe the performance of a control system is in terms of the size of certain signals of interest. The mathematical feature for measuring the size of a signal is the *norm* of a signal. As a transfer function relates different signals, combining it with the norms chosen for the different signals we arrive at the *induced norm* of the transfer function. The induced norm can be used when we want to express specifications such as: the transfer from the class of perturbations that are bounded in magnitude to the output, needs to be bounded by some constant α. In addition, these norm specifications are applied frequently in the design of a *robust controller* [13]. Furthermore, it will also turn out that many specifications in terms of (frequency weighted) bounds on transfer functions are *closed-loop convex*. Three norms and their related induced norms are defined in this section. The material here is based on Chapter 2 of [13].

13.4.1 Norms of signals and systems

Norms of signals

We consider signals mapping $\mathbb{R}$ to $\mathbb{R}$. They are assumed to be piecewise continuous. If we have a signal from $\mathbb{R}^+$ to $\mathbb{R}$, i.e., a signal for which the evolution only starts at time $k=0$, then we extend the signal to a signal from $\mathbb{R}$ to $\mathbb{R}$ by defining the signal to be zero for $k<0$.

We are going to introduce three different norms for such (scalar) signals. First, recall that a norm must have the following four properties:

(i) $\|u\| \geqslant 0$,

(ii) $\|u\| = 0 \Leftrightarrow u(k) = 0, \quad \forall k$,

(iii) $\|au\| = |a|\|u\| \quad \forall a \in \mathbb{R}$,

(iv) $\|u+v\| \leqslant \|u\| + \|v\|$.

The last property is the familiar triangle inequality.

Definition 13.2 *The* **1-norm** *of a signal $u(k)$ is the sum of its modulus:*

$$\|u\|_1 = \sum_{k=-\infty}^{\infty} |u(k)|$$

Definition 13.3 *The* **2-norm** *of a signal $u(k)$ is,*

$$\|u\|_2 = \left(\sum_{k=-\infty}^{\infty} u^2(k) \right)^{\frac{1}{2}}$$

Definition 13.4 *The* **∞-norm** *of a signal $u(k)$ is the least upper bound of its modulus:*

$$\|u\|_\infty = \sup_k |u(k)| \ .$$

Norms of systems

We consider systems that are linear, time-invariant, causal, and (usually) finite dimensional. In the time domain an input-output model for such a system has the form of a convolution equation,

$$y(k) = h(k) * u(k)$$

that is,

$$y(k) = \sum_{n=-\infty}^{\infty} h(k-n)u(n)$$

where h is the impulse response of the system. Causality means that $h(k) = 0$ for $k < 0$. Let H denote the transfer function, i.e., the z-transform of h. Then H is rational (by finite dimensionality) with real coefficients.

Two common norms of the transfer function are defined.

Definition 13.5 *The 2-norm of the transfer function H is defined as*

$$\|H\|_2 = \left(\frac{1}{2\pi} \int_{-\pi}^{\pi} |H(e^{j\omega})|^2 d\omega \right)^{\frac{1}{2}}$$

Definition 13.6 *The ∞-norm of the transfer function H is defined as*

$$\|H\|_\infty = \sup_{|z| \geqslant 1} |H(z)| = \sup_{\substack{\sigma \geqslant 1 \\ \omega \in [0,\pi]}} |H(\sigma e^{j\omega})| \ .$$

For proper stable systems, the ∞-norm will be bounded. Furthermore, the maximum is reached at the unit circle ($\sigma = 1$), and so

Definition 13.7 *The ∞-norm of the stable transfer function H is defined as*

$$\|H\|_\infty = \sup_\omega |H(e^{j\omega})| \ .$$

The ∞-norm of a stable transfer function H equals the distance in the complex plane from the origin to the farthest point on the Nyquist plot of H. It also appears as the peak value on the Bode magnitude plot of H.

Input-output relationships

The question of interest in this section is: If we know how big the input is, how big is the output going to be?

Consider a linear system with input u, output y, and transfer function H, assumed to be stable and strictly proper. Suppose that u is not a fixed signal but that it can be any signal with bounded p-norm $\| u \|_p \leqslant 1$, and suppose we want to say something about the maximum level of y expressed in a m-norm. This quantity is called the p-norm/m-norm *system gain*, and denoted by $\|H\|_{(p,m)}$:

$$\|H\|_{(p,m)} = \sup\{\|y\|_m : \|u\|_p \leqslant 1\}$$

For some values of p and m useful relations between system gains and system norm result: The results are summarized in the following lemma:

Lemma 13.8 *The following relation between system gains and system norms exist:*

$$\| H \|_\infty = \|H\|_{(2,2)} = \sup\{\|y\|_2 : \|u\|_2 \leqslant 1\} \quad (13.2)$$

$$\| H \|_2 = \|H\|_{(2,\infty)} = \sup\{\|y\|_\infty : \|u\|_2 \leqslant 1\} \quad (13.3)$$

$$\| h \|_1 = \|H\|_{(\infty,\infty)} = \sup\{\|y\|_\infty : \|u\|_\infty \leqslant 1\} \quad (13.4)$$

Infinity norm $\|H\|_\infty$:

First we see that $\|H\|_\infty$ is an upper bound on the 2-norm/2-norm system gain:

$$\begin{aligned}\|y\|_2^2 &= \|Y\|_2^2 \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} |H(e^{j\omega})|^2 |U(e^{j\omega})|^2 \, d\omega \\ &\leq \|H\|_\infty^2 \frac{1}{2\pi}\int_{-\pi}^{\pi} |U(e^{j\omega})|^2 \, d\omega \\ &\leq \|H\|_\infty^2 \, \|u\|_2^2\end{aligned}$$

To show that $\|H\|_\infty$ is the least upper bound, first choose a frequency ω_0 where $|H(e^{j\omega})|$ is maximal, that is,

$$|H(e^{j\omega_0})| = \|H\|_\infty$$

Now choose the input u so that

$$|U(e^{j\omega})| = \begin{cases} c & \text{if } |\omega-\omega_0| < \varepsilon \text{ or } |\omega+\omega_0| < \varepsilon\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

where ε is a small positive number and c is chosen so that u has unit 2-norm (i.e., $c = \sqrt{\frac{\pi}{2\varepsilon}}$). Then,

$$\begin{aligned}\|y\|_2^2 &= \frac{1}{2\pi}\int_{-\pi}^{\pi} |H(e^{j\omega})|^2 |U(e^{j\omega})|^2 \, d\omega \\ &= \frac{1}{2\pi}\int_{-\omega_0-\varepsilon}^{-\omega_0+\varepsilon} |H(e^{j\omega})|^2 \frac{\pi}{2\varepsilon} \, d\omega + \frac{1}{2\pi}\int_{\omega_0-\varepsilon}^{\omega_0+\varepsilon} |H(e^{j\omega})|^2 \frac{\pi}{2\varepsilon} \, d\omega \\ &\approx \frac{1}{2\pi}\left[|H(e^{-j\omega_0})|^2\pi + |H(e^{j\omega_0})|^2\pi\right] \\ &\approx |H(e^{j\omega_0})|^2 \\ &\approx \|H\|_\infty^2 \; .\end{aligned}$$

Two norm $\|H\|_2$:
This is an application of the Cauchy-Schwartz inequality:

$$\begin{aligned}
\|y\|_\infty &= \sup_k \left| \sum_{n=-\infty}^{\infty} h(k-n)u(n) \right| \\
&\leqslant \sup_k \left(\sum_{n=-\infty}^{\infty} h^2(k-n) \right)^{\frac{1}{2}} \left(\sum_{n=-\infty}^{\infty} u^2(n) \right)^{\frac{1}{2}} \\
&\leqslant \|h\|_2 \|u\|_2 \\
&\leqslant \|H\|_2 \|u\|_2
\end{aligned}$$

Hence,

$$\|y\|_\infty \leqslant \|H\|_2 \|u\|_2$$

To show that $\|H\|_2$ is the least upper bound, apply the input,

$$u(k) = h(-k)/\|H\|_2$$

Then $\|u\|_2 = 1$ and $|y(0)| = \|H\|_2$, so $\|y\|_\infty \geqslant \|H\|_2$.

One norm $\|h\|_1$:
First, we show that $\|h\|_1$ is an upper bound on the ∞-norm/∞-norm gain:

$$\begin{aligned}
\|y\|_\infty &= \sup_k \left| \sum_{n=-\infty}^{\infty} h(n)u(k-n) \right| \\
&\leqslant \sup_k \sum_{n=-\infty}^{\infty} |h(n)u(k-n)| \\
&\leqslant \sup_k \sum_{n=-\infty}^{\infty} |h(n)|\,|u(k-n)| \\
&\leqslant \sum_{n=-\infty}^{\infty} |h(n)| \|u\|_\infty \\
&\leqslant \|u\|_\infty \|h\|_1
\end{aligned}$$

That $\|h\|_1$ is the least upper bound can be seen as follows. Set $k = 0$ and let

$$u(k-n) := \operatorname{sgn}[h(n)], \quad \forall n.$$

Then, $\|u\|_\infty = 1$ and,

$$\begin{aligned}
y(k) &= \sum_{n=-\infty}^{\infty} h(n)u(k-n) \\
&= \sum_{n=-\infty}^{\infty} |h(n)| \\
&= \|h\|_1
\end{aligned}$$

So $\|y\|_\infty \geqslant \|h\|_1$.

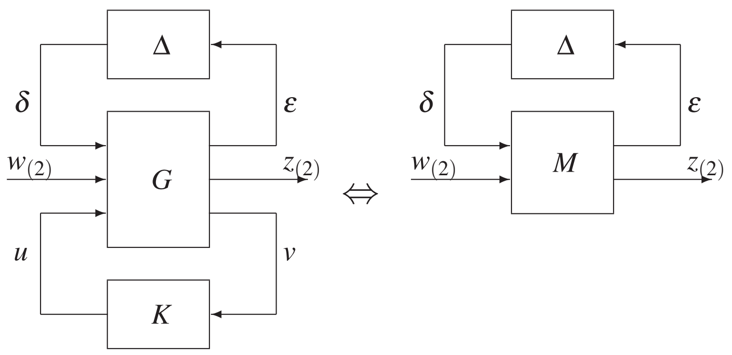


Figure 13.2: General Feedback System with model error

13.5 Robust stability and plant uncertainty

In Section 12.2.1 we raised the question whether a controller K is able to stabilize all perturbed plants in the set $\mathscr{P}$. This question is related to the notion of *robust stability* which is defined below.

Definition 13.9 *Consider the feedback system of Figure 13.2. A controller K provides robust stability if it provides internal stability for all stable model errors with* $\|\Delta\|_\infty < 1$.

For the feedback system of Figure 13.2 we partition $z(k)$ and $w(k)$ as:

$$z(k) = \begin{bmatrix} \varepsilon(k) \\ z_{(2)}(k) \end{bmatrix} \qquad w(k) = \begin{bmatrix} \delta(k) \\ w_{(2)}(k) \end{bmatrix}$$

and we also partition M such that

$$\begin{bmatrix} \varepsilon(k) \\ z_{(2)}(k) \end{bmatrix} = \begin{bmatrix} M_{(1,1)}(q) & M_{(1,2)}(q) \\ M_{(2,1)}(q) & M_{(2,2)}(q) \end{bmatrix} \begin{bmatrix} \delta(k) \\ w_{(2)}(k) \end{bmatrix} .$$

Theorem 13.10 *Assume that the nominal feedback system (so for* $\Delta = 0$*) of Figure 13.2 is internally stable for some controller K. Consider the bounded model error set* $\{\Delta \in \mathscr{S} \mid \|\Delta\|_\infty < 1\}$*. The controller K provides robust stability if and only if*

$$\|M_{(1,1)}\|_\infty \leqslant 1 ,$$

or equivalently, by (12.8), if

$$\left\| \left[G_{(1,1)} + G_{(1,2)} K \left(I - G_{(2,2)} K \right)^{-1} G_{(2,1)} \right]_{(1,1)} \right\|_\infty \leqslant 1$$

or

$$\left\| \left[G_{(1,1)} + G_{(1,2)} Q G_{(2,1)} \right]_{(1,1)} \right\|_\infty \leqslant 1 \ . \tag{13.5}$$

Proof : The closed-loop transfer function for the loop consisting of M and Δ in Figure 13.2 (right) is given by

$$z_{(2)}(k) = T_{\text{cl}}(q)w_{(2)}(k) = \Big(M_{(2,2)}(q) + M_{(2,1)}(q)\Delta(q)\Big(1 - M_{(1,1)}(q)\Delta(q)\Big)^{-1}M_{(1,2)}(q)\Big)w_{(2)}(k)$$

Assume that $\|M_{(1,1)}\|_\infty \leqslant 1$.
($\Leftarrow$) We have a stable function $M_{(1,1)}\Delta$ with

$$\|M_{(1,1)}\Delta\|_\infty \leqslant \|M_{(1,1)}\|_\infty \|\Delta\|_\infty < 1$$

This is equivalent to

$$\|M_{(1,1)}\Delta\|_\infty = \max_{|z|\geqslant 1} |M_{(1,1)}(z)\Delta(z)| < 1$$

We can now derive

$$\begin{aligned}
\|\Big(1 - M_{(1,1)}(q)\Delta(q)\Big)^{-1}\|_\infty &= \max_{|z|\geqslant 1}|\Big(1 - M_{(1,1)}(z)\Delta(z)\Big)^{-1}| \\
&= \Big(\min_{|z|\geqslant 1}|1 - M_{(1,1)}(z)\Delta(z)|\Big)^{-1} \\
&\leqslant \Big(\min_{|z|\geqslant 1} 1 - \max_{|z|\geqslant 1}|M_{(1,1)}(z)\Delta(z)|\Big)^{-1} \\
&< \infty
\end{aligned}$$

which proves that $\Big(1 - M_{(1,1)}\Delta\Big)^{-1}$ is stable. Using the fact that $M_{(2,2)}$, $M_{(2,1)}$, Δ, $M_{(1,2)}$ and $\Big(1 - M_{(1,1)}\Delta\Big)^{-1}$ are stable, we have proved that if $\|M_{(1,1)}\|_\infty \leqslant 1$ then T_{cl} is stable.
($\Rightarrow$) Suppose that $\|M_{(1,1)}\|_\infty > 1$. We will construct an allowable Δ that destabilizes the closed loop system. At some frequency ω^* we have

$$|M_{(1,1)}(e^{j\omega^*})| > 1$$

Now construct a Δ such that

$$\Delta(e^{j\omega^*}) = \Big(M_{(1,1)}(e^{j\omega^*})\Big)^{-1}$$

and for all $0 \leqslant \omega \leqslant \pi$ there holds

$$|\Delta(e^{j\omega})| < 1$$

Then

$$\|\Delta\|_\infty < 1$$

and so we have an allowable Δ. However, we will find that

$$\Big(1 - M_{(1,1)}(e^{j\omega^*})\Delta(e^{j\omega^*})\Big) = 0$$

and therefore $\Big(1 - M_{(1,1)}(q)\Delta(q)\Big)^{-1}$ will be unbounded. This means that the closed loop T_{cl} will not be stable. □

Remark 13.11 It is important to note that (13.5) yields a constraint that is convex in Q. ◇

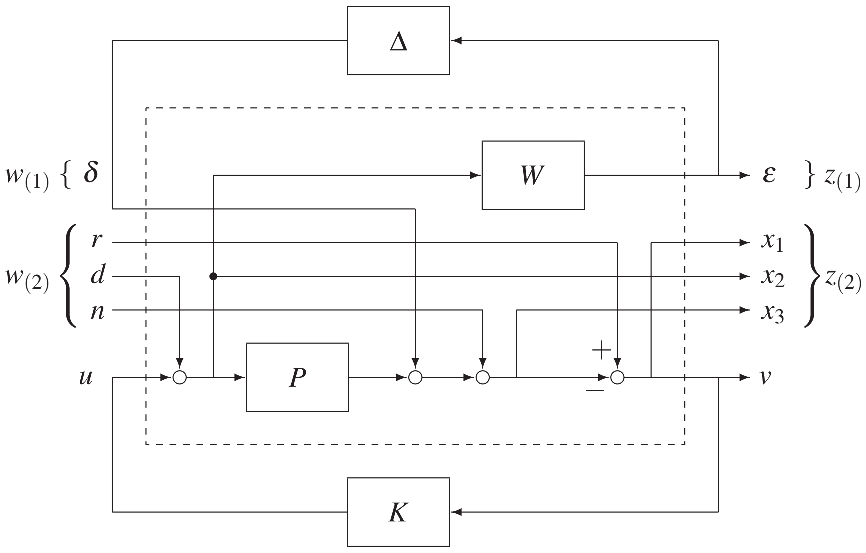


Figure 13.3: Uniform representation for Example 13.12.

Example 13.12 In this example we study the additive model error for the system in Figure 13.3 (Note that compared to the configuration of Figure 12.9 we have $F = 1$).

The relation between input and output signals inside the dashed box of Figure 13.3 is given by:

$$\begin{aligned}
\varepsilon(k) &= 0\cdot\delta(k)+0\cdot r(k)+W\cdot d(k)+0\cdot n(k)+W\cdot u(k) && (13.6)\\
x_1(k) &= -1\cdot\delta(k)+1\cdot r(k)-P\cdot d(k)-1\cdot n(k)-P\cdot u(k) && (13.7)\\
x_2(k) &= 0\cdot\delta(k)+0\cdot r(k)+1\cdot d(k)+0\cdot n(k)+1\cdot u(k) && (13.8)\\
x_3(k) &= 1\cdot\delta(k)+0\cdot r(k)+P\cdot d(k)+1\cdot n(k)+P\cdot u(k) && (13.9)\\
v(k) &= -1\cdot\delta(k)+1\cdot r(k)-P\cdot d(k)-1\cdot n(k)-P\cdot u(k) && (13.10)
\end{aligned}$$

and with $z = \begin{bmatrix} \varepsilon & x_1 & x_2 & x_3 \end{bmatrix}^T$ and $w = \begin{bmatrix} \delta & r & d & n \end{bmatrix}^T$ and

$$\begin{bmatrix} z \\ v \end{bmatrix} = \left[\begin{array}{c|c} G_{(1,1)} & G_{(1,2)} \\ \hline G_{(2,1)} & G_{(2,2)} \end{array}\right] \begin{bmatrix} w \\ u \end{bmatrix}$$

we obtain:

$$\left[\begin{array}{c|c} G_{(1,1)} & G_{(1,2)} \\ \hline G_{(2,1)} & G_{(2,2)} \end{array}\right] = \left[\begin{array}{cccc|c} 0 & 0 & W & 0 & W \\ -1 & 1 & -P & -1 & -P \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & P & 1 & P \\ \hline -1 & 1 & -P & -1 & -P \end{array}\right].$$

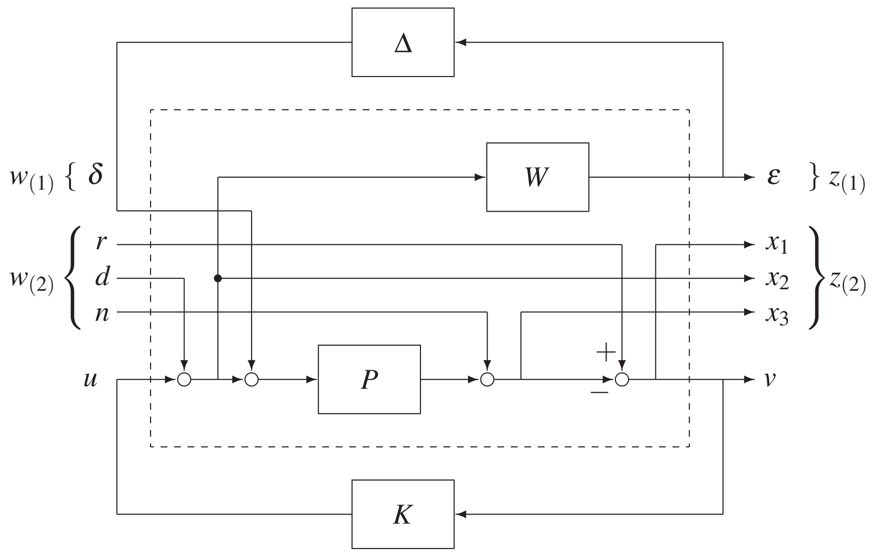


Figure 13.4: Uniform representation for Example 13.13.

The relation between input and output signals including the controller K is given by:

$$\varepsilon(k) = (1+PK)^{-1}\Big(-WK\cdot\delta(k)+WK\cdot r(k)+W\cdot d(k)-WK\cdot n(k)\Big) \tag{13.11}$$

$$x_1(k) = (1+PK)^{-1}\Big(-1\cdot\delta(k)+1\cdot r(k)-P\cdot d(k)-1\cdot n(k)\Big) \tag{13.12}$$

$$x_2(k) = (1+PK)^{-1}\Big(-K\cdot\delta(k)+K\cdot r(k)+1\cdot d(k)-K\cdot n(k)\Big) \tag{13.13}$$

$$x_3(k) = (1+PK)^{-1}\Big(1\cdot\delta(k)+PK\cdot r(k)+P\cdot d(k)+1\cdot n(k)\Big) \tag{13.14}$$

we obtain:

$$M = (1+PK)^{-1}\left[\begin{array}{r|rrr} -WK & WK & W & -WK \\ \hline -1 & 1 & -P & -1 \\ -K & K & 1 & -K \\ 1 & PK & P & 1 \end{array}\right]$$

We see that the robustness criterion for the additive model error now becomes

$$\|M_{(1,1)}\|_\infty = \|-WK(1+PK)^{-1}\|_\infty = \|-WKS\|_\infty < 1 \quad ,$$

where $S=(1+PK)^{-1}$ is called the sensitivity function. The function $KS=K(1+PK)^{-1}$ is sometimes called the control sensitivity function. □

Example 13.13 In this example we study the multiplicative model error for the system in Figure 13.4.

The relation between input and output signals inside the dashed box of Figure 13.4 is given by:

$$\begin{aligned}
\varepsilon(k) &= 0\cdot\delta(k)+0\cdot r(k)+W\cdot d(k)+0\cdot n(k)+W\cdot u(k) && (13.15)\\
x_1(k) &= -P\cdot\delta(k)+1\cdot r(k)-P\cdot d(k)-1\cdot n(k)-P\cdot u(k) && (13.16)\\
x_2(k) &= 0\cdot\delta(k)+0\cdot r(k)+1\cdot d(k)+0\cdot n(k)+1\cdot u(k) && (13.17)\\
x_3(k) &= P\cdot\delta(k)+0\cdot r(k)+P\cdot d(k)+1\cdot n(k)+P\cdot u(k) && (13.18)\\
v(k) &= -P\cdot\delta(k)+1\cdot r(k)-P\cdot d(k)-1\cdot n(k)-P\cdot u(k) && (13.19)
\end{aligned}$$

and with $z = \begin{bmatrix} \varepsilon & x_1 & x_2 & x_3 \end{bmatrix}^T$ and $w = \begin{bmatrix} \delta & r & d & n \end{bmatrix}^T$ and

$$\begin{bmatrix} z \\ v \end{bmatrix} = \left[\begin{array}{c|c} G_{(1,1)} & G_{(1,2)} \\ \hline G_{(2,1)} & G_{(2,2)} \end{array}\right] \begin{bmatrix} w \\ u \end{bmatrix}$$

we obtain:

$$\left[\begin{array}{c|c} G_{(1,1)} & G_{(1,2)} \\ \hline G_{(2,1)} & G_{(2,2)} \end{array}\right] = \left[\begin{array}{cccc|c} 0 & 0 & W & 0 & W \\ -P & 1 & -P & -1 & -P \\ 0 & 0 & 1 & 0 & 1 \\ P & 0 & P & 1 & P \\ \hline -P & 1 & -P & -1 & -P \end{array}\right]$$

The relation between input and output signals including the controller K is given by:

$$\begin{aligned}
\varepsilon(k) &= (1+PK)^{-1}\Big(-WPK\cdot\delta(k)+WK\cdot r(k)+W\cdot d(k)-WK\cdot n(k)\Big) && (13.20)\\
x_1(k) &= (1+PK)^{-1}\Big(-P\cdot\delta(k)+1\cdot r(k)-P\cdot d(k)-1\cdot n(k)\Big) && (13.21)\\
x_2(k) &= (1+PK)^{-1}\Big(-PK\cdot\delta(k)+K\cdot r(k)+1\cdot d(k)-K\cdot n(k)\Big) && (13.22)\\
x_3(k) &= (1+PK)^{-1}\Big(P\cdot\delta(k)+PK\cdot r(k)+P\cdot d(k)+1\cdot n(k)\Big) && (13.23)
\end{aligned}$$

we obtain:

$$M = (1+PK)^{-1}\left[\begin{array}{r|rrr} -WPK & WK & W & -WK \\ \hline -P & 1 & -P & -1 \\ -PK & K & 1 & -K \\ P & PK & P & 1 \end{array}\right]$$

We see that the robustness criterion for the multiplicative model error now becomes

$$\|M_{(1,1)}\|_\infty = \|-WPK(1+PK)^{-1}\|_\infty = \|-WT\|_\infty < 1$$

where $T = PK(1+PK)^{-1}$ is called the complementary sensitivity function.

□

Chapter 14

An Example of Multi-Criteria Controller Design

In this chapter we present a simple multi-criteria controller design example. For a SISO regulator problem, we determine the trade-off curve between the noise sensitivity and stability robustness for plant perturbations. First we determine a controller that minimizes the noise sensitivity when no constraints on robustness are imposed. Such a controller is called an LQG controller. To understand the solvability of a multi-criteria control design, we use the concept of Pareto optimality, which allows us to express the limits of *achievable* performances. Next a convenient way to calculate the noise sensitivity and its gradient is presented. We also explain how the subgradient of the robustness constraint can be obtained. Finally, we calculate the trade-off curve using MATLAB.

14.1 The plant

We consider the design of a regulator for a system with a single actuator (scalar u) which is disturbed by the scalar process noise d. The scalar output we wish to regulate is y and the only signal available to the controller is $y_s = y + n$, where n is sensor noise. A classical block diagram of the closed-loop system is shown in Figure 14.1.

The transfer function of the plant P is given by

$$P(q) = \frac{q^{-1}(1+0.5q^{-1})}{1-1.6q^{-1}+0.7q^{-2}} \tag{14.1}$$

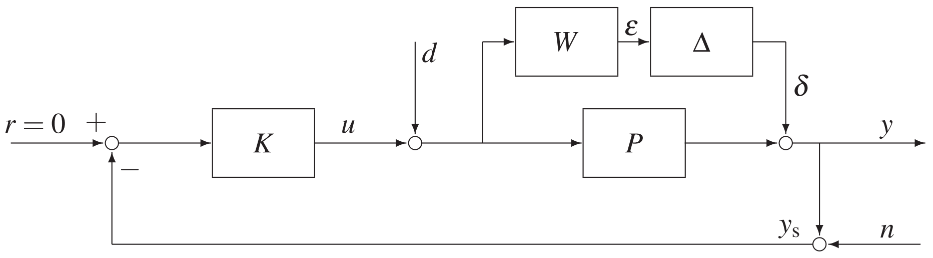


Figure 14.1: Block diagram of the design example.

It is assumed that the process noise d and sensor noise n are zero mean white noise sequences with RMS values given by:

$$\mathrm{RMS}\Big(d(k)\Big) \overset{\text{def}}{=} \sqrt{E\Big[d^2(k)\Big]} = 4$$
$$\mathrm{RMS}\Big(n(k)\Big) \overset{\text{def}}{=} \sqrt{E\Big[n^2(k)\Big]} = 1$$

where $E[\cdot]$ denotes statistical expected value.

14.2 Engineering specifications

The specifications we will explore are:

- *Plant output regulation* — a maximum RMS response at the plant output y due to the process and sensor noises:

$$\mathrm{RMS}(y) \quad = \quad \sqrt{E[y^2(k)]} \leqslant \alpha$$

- *Actuator effort* — a maximum RMS response at the actuator u due to the process and sensor noises.

$$\mathrm{RMS}(u) \quad = \quad \sqrt{E[u^2(k)]} \leqslant \beta$$

- *Robustness to additive plant perturbations* — a minimum robustness of the closed-loop system to additive plant perturbations. We require that our closed-loop system remains stable if P is perturbed to $P + W\Delta$, where W is a filter transfer function and Δ is any stable transfer function with $||\Delta||_\infty < 1$.

14.3 General formulation of the basic feedback loop

A general way to represent the basic feedback loop represented in Figure 12.7, such that it represents a wide class of controller design problems (even for MIMO systems) is given next. For this purpose, we redraw Figure 12.7 as Figure 14.2, where we also take an additive model error structure into account. If we now define the following vector quantities:

$$w(k) = \begin{bmatrix} \delta(k) \\ r(k) \\ d(k) \\ n(k) \end{bmatrix} \quad z(k) = \begin{bmatrix} \varepsilon(k) \\ u(k) \\ y(k) \end{bmatrix} \quad v(k) = r(k) - n(k) - y(k) \ . \tag{14.2}$$

The relation between input and output signals including the controller K is given by:

$$\begin{aligned} \varepsilon(k) &= (1+PK)^{-1}\Big(-WK\cdot\delta(k) + WK\cdot r(k) + W\cdot d(k) - WK\cdot n(k)\Big) \\ u(k) &= (1+PK)^{-1}\Big(-K\cdot\delta(k) + K\cdot r(k) - PK\cdot d(k) - K\cdot n(k)\Big) \\ y(k) &= (1+PK)^{-1}\Big(1\cdot\delta(k) + PK\cdot r(k) + P\cdot d(k) - PK\cdot n(k)\Big) \ . \end{aligned}$$

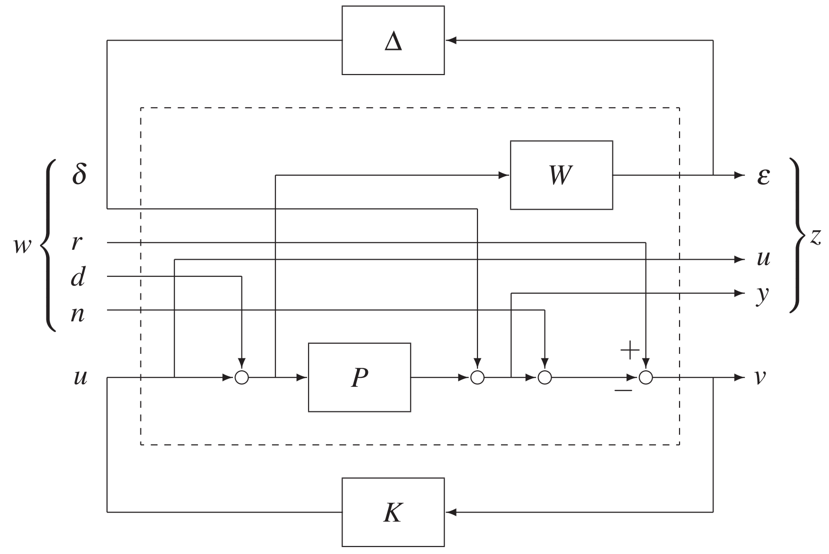


Figure 14.2: Uniform representation for the design example.

So we obtain:

$$\left[\begin{array}{c|ccc} M_{(1,1)} & M_{(1,2)} & M_{(1,3)} & M_{(1,4)} \\ \hline M_{(2,1)} & M_{(2,2)} & M_{(2,3)} & M_{(2,4)} \\ M_{(3,1)} & M_{(3,2)} & M_{(3,3)} & M_{(3,4)} \end{array}\right] = (1+PK)^{-1} \left[\begin{array}{c|ccc} -WK & WK & W & -WK \\ \hline -K & K & -PK & -K \\ 1 & PK & P & -PK \end{array}\right].$$

Now we introduce the stabilizing controller K as introduced in Theorem 12.5. Using the fact that $G_{(2,2)} = -P$, we obtain:

$$K = Q(1-PQ)^{-1} \tag{14.3}$$

where $Q \in \mathscr{S}$. Now we use the property that $K(1+PK)^{-1} = Q$ and $(1+PK)^{-1} = 1-PQ$ and we obtain:

$$\left[\begin{array}{c|ccc} M_{(1,1)} & M_{(1,2)} & M_{(1,3)} & M_{(1,4)} \\ \hline M_{(2,1)} & M_{(2,2)} & M_{(2,3)} & M_{(2,4)} \\ M_{(3,1)} & M_{(3,2)} & M_{(3,3)} & M_{(3,4)} \end{array}\right] = \left[\begin{array}{c|ccc} -WQ & WQ & W-WPQ & -WQ \\ \hline -Q & Q & -PQ & -Q \\ 1-PQ & PQ & P(1-PQ) & -PQ \end{array}\right] \tag{14.4}$$

Clearly, the transfer function M is stable as long as $Q \in \mathscr{S}$.

14.4 Linear Quadratic Gaussian design

The Linear Quadratic Gaussian (LQG) controller design problem minimizes in an analytic manner (over all stabilizing controllers) the following weighted cost function,

$$J = E[y^2(k) + \rho u^2(k)] = \mathrm{RMS}^2\Big(y(k)\Big) + \rho\,\mathrm{RMS}^2\Big(u(k)\Big) \tag{14.5}$$

for a fixed scalar positive number ρ. The philosophy behind (14.5) is the following. In order to keep the plant output y close to zero, a control signal u having a certain amount of energy is required.

Intuitively, we would expect that the control energy increases when the plant output energy decreases. Nevertheless, we actually want both these energies to be small. In other words, we want to minimize the function J where ρ is a parameter which determines the trade-off between a small plant output y and a small control signal u. The function J will be referred to as the *noise sensitivity*. As has already been discussed in this section, the controller that minimizes J for a certain value of ρ is the *LQG controller*. We will now take a closer look at this particular controller.

The LQG controller consists of two parts. The first part estimates the state of the plant in the presence of white Gaussian process and measurement noise. This is an *optimal state observer* or *Kalman filter*. The second part is a constant state feedback gain, which is chosen such that the quadratic cost function (14.5) is minimized. This part is called an LQ controller. The observer as well as the LQ controller can be found analytically. However, the derivations are beyond the scope of these notes. They can, for example, be found in [36] and [39]. Below only the final results are stated.

Let $(A_\mathrm{p}, B_\mathrm{p}, C_\mathrm{p}, D_\mathrm{p})$ be the system matrices of the plant P. The state space description of the LQG controller is given by

$$\begin{aligned}
A_\mathrm{c} &= A_\mathrm{p} - A_\mathrm{p} L C_\mathrm{p} - (B_\mathrm{p} - A_\mathrm{p} L D_\mathrm{p})(I + F L D_\mathrm{p})^{-1}(F - F L C_\mathrm{p}) \\
B_\mathrm{c} &= A_\mathrm{p} L - (B_\mathrm{p} - A_\mathrm{p} L D_\mathrm{p})(I + F L D_\mathrm{p})^{-1} F L \\
C_\mathrm{c} &= F - F L C_\mathrm{p} + F L D_\mathrm{p}(I + F L D_\mathrm{p})^{-1}(F - F L C_\mathrm{p}) \\
D_\mathrm{c} &= F L + F L D_\mathrm{p}(I + F L D_\mathrm{p})^{-1} F L
\end{aligned}$$

where F is the feedback gain of the LQ controller and L the Kalman gain of the state observer.

The LQG controller can easily be calculated using the *MATLAB Control Toolbox*: the function `dlqry` calculates F, the function `dlqe` calculates L, and the function `dreg` calculates the system matrices of the controller.

```
F=dlqry(Ap,Bp,Cp,Dp,1,rho); L=dlqe(Ap,Bp,Cp,RMS2_d,RMS2_n);
[Ac,Bc,Cc,Dc]=dreg(Ap,Bp,Cp,Dp,F,L);
```

where `RMS2_d` and `RMS2_n` correspond to $\mathrm{RMS}^2\Big(d(k)\Big)$ and $\mathrm{RMS}^2\Big(n(k)\Big)$, respectively.

14.5 Example formulation of a robust controller design problem

We have seen in the previous section that the LQG controller is an optimal controller with respect to the noise sensitivity. The LQG controller is, however, not very robust with respect to plant uncertainty. In the remainder of this chapter we will investigate the trade-off between a good noise sensitivity and robustness to additive plant perturbations.

If we consider $W(e^{j\omega})$, defined in Section 14.2, to be equal to D_min for all frequencies ω, and $\|\Delta\|_\infty < 1$, the controller K is robust with respect to additive plant perturbations if $\|D_\mathrm{min} KS\|_\infty = \|D_\mathrm{min} Q\|_\infty \leqslant 1$ or equivalently if $\|Q\|_\infty \leqslant \dfrac{1}{D_\mathrm{min}}$. Note that we consider an additive perturbation deviation $W\Delta = D_\mathrm{min}\Delta$ on P where Δ is bounded: $\|\Delta\|_\infty < 1$. So if D_min increases, we allow a larger perturbation, and if the controller can still deal with that, it is thus more robust. Hence, a larger D_min implies a controller which is more robust.

We want to design a controller Q satisfying the following engineering specifications:

1. Minimal noise sensitivity: $J = \mathrm{RMS}^2\Big(y(k)\Big) + \rho\,\mathrm{RMS}^2\Big(u(k)\Big) \le J_\mathrm{max}$, $\rho = 10^{-4}$

2. Robustness to additive plant perturbations: $\|D_\mathrm{min} Q\|_\infty \le 1$

Obviously, the controlled system must also be stable. To assure stability, the controller is parametrized using the Q parametrization, where Q is chosen to have the following structure

$$Q(q) = \frac{\theta_0 + \theta_1 q^{-1} + \theta_2 q^{-2} + \theta_3 q^{-3} + \theta_4 q^{-4}}{1 + p_1 q^{-1} + p_2 q^{-2} + p_3 q^{-3} + p_4 q^{-4}}$$

with all roots of $1 + p_1 q^{-1} + p_2 q^{-2} + p_3 q^{-3} + p_4 q^{-4}$ strictly inside the unit circle.

For the LQG controller with ρ equal to 10^{-4}, $D_{\min}$ equals 0.049. Now we want to calculate controllers which are more robust, so for which $D_{\min} \geq 0.049$. By making the bound $\|KS\|_\infty = \|Q\|_\infty \leq 1/D_{\min}$ tighter and tighter, that is increasing $D_{\min}$, and solving the corresponding constrained optimization problems, we can calculate the desired trade-off curve. Note that this is a nonlinear optimization problem, since J and $D_{\min}$ are nonlinear in the parameters $\theta_0, \ldots, \theta_4$ and $p_1, \ldots, p_4$. To facilitate the optimization problem, we fix the poles of Q and only allow its zeros to change. We determine Q when K is an LQG controller with $\rho = 10^{-4}$. Then, we take p_1, p_2, p_3 and p_4 equal to their values in the LQG controller. Then we can formulate the determination of $\theta_0, \theta_1, \theta_2$, θ_3 and θ_4 according to the engineering specs as a convex optimization problem with one constraint.

14.6 Computing the noise sensitivity and its gradient

For ease of notation, the following definitions are made

$$\bar{z}(k) = \begin{bmatrix} u(k) \\ y(k) \end{bmatrix}, \qquad \bar{w}(k) = \begin{bmatrix} d(k) \\ n(k) \end{bmatrix}, \qquad T = \begin{bmatrix} M_{(2,3)} & M_{(2,4)} \\ M_{(3,3)} & M_{(3,4)} \end{bmatrix} \tag{14.6}$$

Obviously in the nominal case ($\Delta = 0$ and so $\delta = 0$) and no reference signal ($r(k) = 0$) we find $\bar{z}(k) = T(q)\bar{w}(k)$.

Recall that the noise sensitivity J is given by

$$J = \text{RMS}^2\Big(y(k)\Big) + \rho\,\text{RMS}^2\Big(u(k)\Big)$$

Hence, in order to determine J the RMS values of $y(k)$ and $u(k)$ have to be calculated. In Section 14.2 the following expressions are given for these RMS values

$$\begin{aligned} \text{RMS}^2\Big(u(k)\Big) &= \frac{E[d^2(k)]}{2\pi}\int_{-\pi}^{\pi} |M_{(2,3)}(e^{j\omega})|^2\, d\omega + \frac{E[n^2(k)]}{2\pi}\int_{-\pi}^{\pi} |M_{(2,4)}(e^{j\omega})|^2\, d\omega \\ \text{RMS}^2\Big(y(k)\Big) &= \frac{E[d^2(k)]}{2\pi}\int_{-\pi}^{\pi} |M_{(3,3)}(e^{j\omega})|^2\, d\omega + \frac{E[n^2(k)]}{2\pi}\int_{-\pi}^{\pi} |M_{(3,4)}(e^{j\omega})|^2\, d\omega \ . \end{aligned}$$

Although theoretically correct, these expressions are not very useful in practice. It is not easy to find an analytic expression for these integrals. Numerical integration is also not a good choice, since it has to be performed in each iteration step of the optimization process. A better way to calculate the RMS values is by solving a Lyapunov equation which originates from a state space representation of the controlled system. This approach will be highlighted below. First, we explain how a state space representation of the controlled system can be obtained from state space descriptions of P and Q. Next, we point out how the Lyapunov equation can be obtained, and how it can be solved. Finally, we compute the gradient of the noise sensitivity.

14.6.1 State space representation of the controlled system

Combining equations (14.4) and (14.6) we find

$$\begin{aligned} T &= \begin{bmatrix} M_{(2,3)} & M_{(2,4)} \\ M_{(3,3)} & M_{(3,4)} \end{bmatrix} \\ &= \frac{-1}{1+PK} \begin{bmatrix} PK & K \\ -P & PK \end{bmatrix} \\ &= \begin{bmatrix} -PQ & -Q \\ P(1-PQ) & -PQ \end{bmatrix} \end{aligned} \tag{14.7}$$

A state space representation of this system T can be written as

$$x(k+1) = A_\mathrm{t} x(k) + B_\mathrm{t} \bar{w}(k) \tag{14.8}$$

$$\bar{z}(k) = C_\mathrm{t} x(k) + D_\mathrm{t} \bar{w}(k) \tag{14.9}$$

In this section we will derive expression for the matrices A_t, B_t, C_t, and D_t.

Let $(A_\mathrm{p}, B_\mathrm{p}, C_\mathrm{p}, D_\mathrm{p})$ be the system matrices of the plant P and let $(A_\mathrm{q}, B_\mathrm{q}, C_\mathrm{q}, D_\mathrm{q})$ be the system matrices of Q. In short-hand notation:

$$P \sim \left[\begin{array}{c|c} A_\mathrm{p} & B_\mathrm{p} \\ \hline C_\mathrm{p} & D_\mathrm{p} \end{array}\right], \qquad Q \sim \left[\begin{array}{c|c} A_\mathrm{q} & B_\mathrm{q} \\ \hline C_\mathrm{q} & D_\mathrm{q} \end{array}\right]$$

The goal is to express the system matrices $(A_\mathrm{t}, B_\mathrm{t}, C_\mathrm{t}, D_\mathrm{t})$ in terms of $(A_\mathrm{p}, B_\mathrm{p}, C_\mathrm{p}, D_\mathrm{p})$ and $(A_\mathrm{q}, B_\mathrm{q}, C_\mathrm{q}, D_\mathrm{q})$. The key to this is the state space representation of a series connection of two systems.

Assume we have two systems: $y_1(k) = G_1 u_1(k)$ and $y_2(k) = G_2 u_2(k)$. The state space representation of these systems is given by

$$\begin{aligned} x_1(k+1) &= A_1 x_1(k) + B_1 u_1(k) & \qquad x_2(k+1) &= A_2 x_2(k) + B_2 u_2(k) \\ y_1(k) &= C_1 x_1(k) + D_1 u_1(k) & y_2(k) &= C_2 x_2(k) + D_2 u_2(k) \end{aligned}$$

It is easy to see that the series connection of these systems, $y_1(k) = G_1 G_2 u_2(k)$ obtained by setting $y_2(k) = u_1(k)$ has a state space representation given by:

$$\begin{aligned} \begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} &= \begin{bmatrix} A_1 & B_1 C_2 \\ 0 & A_2 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} B_1 D_2 \\ B_2 \end{bmatrix} u_2(k) \\ y_1(k) &= \begin{bmatrix} C_1 & D_1 C_2 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + D_1 D_2 u_2(k) \end{aligned}$$

or in short-hand notation

$$G_1 G_2 \sim \left[\begin{array}{cc|c} A_1 & B_1 C_2 & B_1 D_2 \\ 0 & A_2 & B_2 \\ \hline C_1 & D_1 C_2 & D_1 D_2 \end{array}\right]$$

We rewrite T as

$$T = \begin{bmatrix} -QP & -Q \\ P(1-QP) & -QP \end{bmatrix} = \begin{bmatrix} -Q \\ 1-QP \end{bmatrix} \begin{bmatrix} P & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{14.10}$$

Now it is easy to obtain the following state space representations.

$$QP \sim \left[\begin{array}{cc|c} A_\mathrm{q} & B_\mathrm{q}C_\mathrm{p} & B_\mathrm{q}D_\mathrm{p} \\ 0 & A_\mathrm{p} & B_\mathrm{p} \\ \hline C_\mathrm{q} & D_\mathrm{q}C_\mathrm{p} & D_\mathrm{q}D_\mathrm{p} \end{array}\right]$$

$$1-QP \sim \left[\begin{array}{cc|c} A_\mathrm{q} & B_\mathrm{q}C_\mathrm{p} & -B_\mathrm{q}D_\mathrm{p} \\ 0 & A_\mathrm{p} & -B_\mathrm{p} \\ \hline C_\mathrm{q} & D_\mathrm{q}C_\mathrm{p} & 1-D_\mathrm{q}D_\mathrm{p} \end{array}\right]$$

$$\begin{bmatrix} -Q \\ 1-QP \end{bmatrix} \sim \left[\begin{array}{ccc|c} A_\mathrm{q} & 0 & 0 & -B_\mathrm{q} \\ 0 & A_\mathrm{q} & B_\mathrm{q}C_\mathrm{p} & -B_\mathrm{q}D_\mathrm{p} \\ 0 & 0 & A_\mathrm{p} & -B_\mathrm{p} \\ \hline C_\mathrm{q} & 0 & 0 & -D_\mathrm{q} \\ 0 & C_\mathrm{q} & D_\mathrm{q}C_\mathrm{p} & 1-D_\mathrm{q}D_\mathrm{p} \end{array}\right]$$

$$\begin{bmatrix} P & 1 \end{bmatrix} \sim \left[\begin{array}{c|cc} A_\mathrm{p} & B_\mathrm{p} & 0 \\ \hline C_\mathrm{p} & D_\mathrm{p} & 1 \end{array}\right]$$

Combining these representations according to equation (14.10), we get

$$T \sim \left[\begin{array}{c|c} A_\mathrm{t} & B_\mathrm{t} \\ \hline C_\mathrm{t} & D_\mathrm{t} \end{array}\right] = \left[\begin{array}{cccc|cc} A_\mathrm{q} & 0 & 0 & -B_\mathrm{q}C_\mathrm{p} & -B_\mathrm{q}D_\mathrm{p} & -B_\mathrm{q} \\ 0 & A_\mathrm{q} & B_\mathrm{q}C_\mathrm{p} & -B_\mathrm{q}D_\mathrm{p}C_\mathrm{p} & -B_\mathrm{q}D_\mathrm{p}^2 & -B_\mathrm{q}D_\mathrm{p} \\ 0 & 0 & A_\mathrm{p} & -B_\mathrm{p}C_\mathrm{p} & -B_\mathrm{p}D_\mathrm{p} & -B_\mathrm{p} \\ 0 & 0 & 0 & A_\mathrm{p} & B_\mathrm{p} & 0 \\ \hline C_\mathrm{q} & 0 & 0 & -D_\mathrm{q}C_\mathrm{p} & -D_\mathrm{q}D_\mathrm{p} & -D_\mathrm{q} \\ 0 & C_\mathrm{q} & D_\mathrm{q}C_\mathrm{p} & C_\mathrm{p}-D_\mathrm{q}D_\mathrm{p}C_\mathrm{p} & D_\mathrm{p}-D_\mathrm{q}D_\mathrm{p}^2 & -D_\mathrm{q}D_\mathrm{p} \end{array}\right] \tag{14.11}$$

Note that $(A_\mathrm{t}, B_\mathrm{t}, C_\mathrm{t}, D_\mathrm{t})$ is not a minimal realization.

14.6.2 The Lyapunov equation

The RMS values of $y(k)$ and $u(k)$ can be calculated from the expected value of $\bar{z}(k)\bar{z}^T(k)$, because

$$E\Big[\bar{z}(k)\bar{z}^T(k)\Big] = \begin{bmatrix} \mathrm{RMS}^2\Big(u(k)\Big) & \star \\ \star & \mathrm{RMS}^2\Big(y(k)\Big) \end{bmatrix}, \tag{14.12}$$

where $\star$ is used to denote entries which we are not interested in. Using (14.9) we get

$$\begin{aligned} E\Big[\bar{z}(k)\bar{z}^T(k)\Big] &= E\Big[\Big(C_\mathrm{t}x(k)+D_\mathrm{t}\bar{w}(k)\Big)\Big(C_\mathrm{t}x(k)+D_\mathrm{t}\bar{w}(k)\Big)^T\Big] \\ &= C_\mathrm{t}E\Big[x(k)x^T(k)\Big]C_\mathrm{t}^T + D_\mathrm{t}E\Big[\bar{w}(k)\bar{w}^T(k)\Big]D_\mathrm{t}^T \end{aligned}$$

where use was made of the statistical independence of $x(k)$ and $\bar{w}(k)$, that is

$$E\Big[x(k)\bar{w}^T(k)\Big] = 0$$

In a similar way (14.8) gives

$$E\Big[x(k+1)x^T(k+1)\Big] = A_\mathrm{t}E\Big[x(k)x^T(k)\Big]A_\mathrm{t}^T + B_\mathrm{t}E\Big[\bar{w}(k)\bar{w}^T(k)\Big]B_\mathrm{t}^T \tag{14.13}$$

For short-hand notation we define[1]

$$X = E\Big[x(k+1)x^T(k+1)\Big] = E\Big[x(k)x^T(k)\Big]$$
$$W = E\Big[\bar{w}(k)\bar{w}^T(k)\Big] = \begin{bmatrix} \mathrm{RMS}^2\Big(d(k)\Big) & 0 \\ 0 & \mathrm{RMS}^2\Big(n(k)\Big) \end{bmatrix} .$$

With these definitions (14.13) can be written as

$$X = A_t X A_t^T + B_t W B_t^T \tag{14.14}$$

This equation is a famous one. It is often called the discrete Lyapunov equation. Note that this equation is linear in X. The solution X to this equation can easily be calculated using the MATLAB function `dlyap`.

```
X=dlyap(At,Bt*W*Bt')
```

Now we define

$$C_t = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}, \qquad D_t = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \tag{14.15}$$

With these definitions we can write

$$C_t X C_t^T + D_t W D_t^T = \begin{bmatrix} c_1 X c_1^T + d_1 W d_1^T & c_1 X c_2^T + d_1 W d_2^T \\ c_2 X c_1^T + d_2 W d_1^T & c_2 X c_2^T + d_2 W d_2^T \end{bmatrix}$$

Using (14.12) the noise sensitivity function can be computed as

$$J = (c_2 X c_2^T + d_2 W d_2^T) + \rho(c_1 X c_1^T + d_1 W d_1^T) \tag{14.16}$$

where X is the solution of (14.14).

14.6.3 The gradient of the noise sensitivity

To compute the gradient, we have to differentiate equation (14.16) with respect to the parameters θ_0, θ_1, θ_2, θ_3, and θ_4. In general, only the matrix W will not depend on these parameters. However, if we represent Q in the controller canonical form, then also the matrix X does not depend on the parameters. This is due to the fact that in this form, the matrices A_q and B_q are independent of the parameters θ_0, θ_1, θ_2, θ_3, and θ_4. The controller canonical form of Q is given by:

$$A_q = \begin{bmatrix} -p_1 & -p_2 & -p_3 & -p_4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \qquad B_q = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

$$C_q = \begin{bmatrix} \theta_1 - \theta_0 p_1 & \theta_2 - \theta_0 p_2 & \theta_3 - \theta_0 p_3 & \theta_4 - \theta_0 p_4 \end{bmatrix}, \qquad D_q = \theta_0 .$$

[1]Note that the matrix W defined here is not related in any way to the filter transfer function W used previously.

Define the parameter vector by $\theta = [\,\theta_0\ \theta_1\ \theta_2\ \theta_3\ \theta_4\,]$. By partial differentiation we can write the gradient of the noise sensitivity as

$$\begin{aligned}\frac{\partial J}{\partial \theta} &= \frac{\partial c_2}{\partial \theta}\frac{\partial}{\partial c_2}\left(c_2 X c_2^T\right) + \frac{\partial d_2}{\partial \theta}\frac{\partial}{\partial d_2}\left(d_2 W d_2^T\right) + \rho\frac{\partial c_1}{\partial \theta}\frac{\partial}{\partial c_1}\left(c_1 X c_1^T\right) + \rho\frac{\partial d_1}{\partial \theta}\frac{\partial}{\partial d_1}\left(d_1 W d_1^T\right) \\ &= 2\left(\frac{\partial c_2}{\partial \theta} X c_2^T + \frac{\partial d_2}{\partial \theta} W d_2^T + \rho\frac{\partial c_1}{\partial \theta} X c_1^T + \rho\frac{\partial d_1}{\partial \theta} W d_1^T\right) \qquad (14.17)\end{aligned}$$

It is straightforward to obtain the necessary partial derivatives by looking at equations (14.15) and (14.11).

$$\frac{\partial c_1}{\partial \theta} = \begin{bmatrix} -p_1 & -p_2 & -p_3 & -p_4 & 0 & \cdots & 0 & -C_\mathrm{p} \\ 1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

$$\frac{\partial c_2}{\partial \theta} = \begin{bmatrix} 0 & \cdots & 0 & -p_1 & -p_2 & -p_3 & -p_4 & C_\mathrm{p} & -D_\mathrm{p}C_\mathrm{p} \\ 0 & \cdots & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\frac{\partial d_1}{\partial \theta} = \begin{bmatrix} -D_\mathrm{p} & -1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\frac{\partial d_2}{\partial \theta} = \begin{bmatrix} -D_\mathrm{p}^2 & -D_\mathrm{p} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

14.7 Computing the robustness constraint and its subgradient

Consider the transfer function $G(e^{j\omega})$ that is affine in the parameter vector θ:

$$G(e^{j\omega}) = G_0(e^{j\omega}) + G_1^T(e^{j\omega})\theta \ . \qquad (14.18)$$

Recall that the induced ∞-norm of a transfer function equals the maximum of its magnitude, that is

$$||G||_\infty = \max_\omega |G(e^{j\omega})|$$

How the ∞-norm can be computed is discussed in Section 13.4.1.

How to compute the ∞-norm and its gradient? This requires a search. Set up a fine grid of frequency points,

$$\{\omega_1, \omega_2, \cdots, \omega_N\}$$

Then an estimate for $\|G\|_\infty$ is

$$\max_{1\leqslant m\leqslant N} |G(e^{j\omega_m})|$$

Alternatively, one could find where $|G(e^{j\omega})|$ is maximal by solving the equation,

$$\frac{d|G(e^{j\omega})|^2}{d\omega} = 0$$

This derivative can be computed in closed form because G is rational. It then remains to compute the roots of a polynomial. Next we focus on the computation of the subgradient. Using a reasoning that is similar to the one that is used to determine the gradient of the function $f_\infty(\theta) = \|F_0(n) + F_1^T(n)\theta\|_\infty$ in Section 6.2, it can be shown that the subgradient of $G(e^{j\omega})$ is given by

$$\nabla_\theta \|G\|_\infty = \mathrm{Re}\left\{ \frac{|G(e^{j\omega_0})|}{G(e^{j\omega_0})} G_1(e^{j\omega_0}) \right\} \tag{14.19}$$

where

$$\omega_0 = \arg\max_\omega |G(e^{j\omega})| \quad .$$

Recall that the robustness constraint to be considered is

$$\|Q\|_\infty \leqslant \frac{1}{D_{\min}} \quad .$$

Now we use expression (14.19) to calculate the subgradient of the ∞-norm of $Q(e^{j\omega})$. Let

$$Q(e^{j\omega}) = \frac{Q_{\mathrm{num}}(e^{j\omega})}{Q_{\mathrm{den}}(e^{j\omega})} \tag{14.20}$$

where Q_{num} and Q_{den} are the numerator and denominator polynomial of the transfer function Q. We have

$$\begin{aligned} Q_{\mathrm{num}}(e^{j\omega}) &= \theta_0 + \theta_1 e^{-j\omega} + \theta_2 e^{-j2\omega} + \theta_3 e^{-j3\omega} + \theta_4 e^{-j4\omega} \\ &= \begin{bmatrix} 1 & e^{-j\omega} & e^{-j2\omega} & e^{-j3\omega} & e^{-j4\omega} \end{bmatrix} \theta \,. \end{aligned} \tag{14.21}$$

If we equate G and Q, and if we compare (14.18) with (14.20)–(14.21) we get

$$G_0(e^{j\omega}) = 0 \quad \text{and} \quad G_1^T(e^{j\omega}) = \frac{1}{Q_{\mathrm{den}}(e^{j\omega})} \begin{bmatrix} 1 & e^{-j\omega} & e^{-j2\omega} & e^{-j3\omega} & e^{-j4\omega} \end{bmatrix} \quad .$$

Therefore, the subgradient of $\|Q\|_\infty$ is given by

$$\nabla_\theta \|Q\|_\infty = \mathrm{Re}\left\{ \frac{|Q(e^{j\omega_{(1,1)}})|}{Q(e^{j\omega_{(1,1)}})} \left(\frac{1}{Q_{\mathrm{den}}(e^{j\omega_{(1,1)}})} \begin{bmatrix} 1 \\ e^{-j\omega_{(1,1)}} \\ e^{-j2\omega_{(1,1)}} \\ e^{-j3\omega_{(1,1)}} \\ e^{-j4\omega_{(1,1)}} \end{bmatrix} \right) \right\} \tag{14.22}$$

$$= \mathrm{Re}\left\{ \frac{|Q(e^{j\omega_{(1,1)}})|}{Q_{\mathrm{num}}(e^{j\omega_{(1,1)}})} \begin{bmatrix} 1 \\ e^{-j\omega_{(1,1)}} \\ e^{-j2\omega_{(1,1)}} \\ e^{-j3\omega_{(1,1)}} \\ e^{-j4\omega_{(1,1)}} \end{bmatrix} \right\} \tag{14.23}$$

where

$$\omega_{(1,1)} = \arg\max_\omega |Q(e^{j\omega})| \quad .$$

14.8 MATLAB implementation

The previous sections have given us all the necessary ingredients to write a MATLAB program which calculates the trade-off curve between noise sensitivity and robustness for additive plant perturbations for the controller structure discussed in Section 14.5.

We write a program called `mcd.m` which initializes the optimization procedure and calculates several points of the trade-off curve. This program uses two MATLAB functions: `mcd_J.m` and `mcd_con.m`. The function `mcd_J.m` evaluates the noise sensitivity J and its gradient. The function `mcd_con.m` calculates the ∞-norm of $M_{(1,1)}$ and its subgradient. The MATLAB code of these three programs is given below.

mcd

At the beginning of the program some variables are declared global, because these variables will also be used by the function `mcd_J`. First, the LQG controller for $\rho = 10^{-4}$ is computed. From this controller, Q is determined. Next, the Lyapunov equation (14.14) is solved. Note that we only have to solve this equation once, since according to (14.11), the system matrices A_t and B_t of the closed-loop system do not depend on the parameters $\theta_0, \ldots, \theta_4$. Then, we set some tolerances for the optimization procedure using the `options` vector (see also Chapter 9). The values of the parameters $\theta_0, \ldots, \theta_4$ calculated from the LQG controller are used as an initial guess. Finally, we start a sequence of optimization procedures using the function `fmincon` from the optimization toolbox (see also Section 9.4). During this sequence, the bound on the ∞-norm of $M_{(1,1)}$ is made tighter. The values of the parameters calculated during the ith optimization are used as initial guess for the parameters in the $(i+1)$th optimization. The function `fmincon` uses the Sequential Quadratic Programming algorithm to solve the optimization problem. It seems to be all right to use this algorithm, because the noise sensitivity is quadratic in the parameters (see equation (14.16)). Alternatively, we could use a convex optimization method, for example the cutting plane algorithm (see Section 6.4). At the end of the next section we will make some comments on this. The values of J, $\|M_{(1,1)}\|_\infty$, and $\theta_0, \ldots, \theta_4$ are stored in the matrix `results`. The total number of function evaluations and the total number of gradient evaluations are counted.

```
% MCD   Multicriteria controller design

RMS2_d=16;               % Squared RMS process noise.
RMS2_n=1;                % Squared RMS sensor noise.
rho=1e-4;
num_p=[0 1 .5];          % Numerator polynomial of the plant.
den_p=[1 -1.6 .7];       % Denomerator polynomial of the plant.

% State space representation of P.
[Ap,Bp,Cp,Dp]=tf2ss(num_p,den_p);

% Initial estimate LQG controller
F=dlqry(Ap,Bp,Cp,Dp,1,rho);            % Feedback gain.
L=dlqe(Ap,Bp,Cp,RMS2_d,RMS2_n);        % Kalman gain.
[Ac,Bc,Cc,Dc]=dreg(Ap,Bp,Cp,Dp,F,L);   % Compute controller.
[num_lqg,den_lqg]=ss2tf(Ac,Bc,Cc,Dc); % Convert to transfer function.
num_q=conv(num_lqg,den_p);
den_q=conv(den_lqg,den_p)+conv(num_lqg,num_p);
```

```
% State space representation of Q.
[Aq,Bq,Cq,Dq]=tf2ss(num_q,den_q);

% Solve Lyapunov equation.
At = [ Aq, zeros(4,6), -Bq*Cp
       zeros(4,4), Aq, Bq*Cp, -Bq*Dp*Cp
       zeros(2,8), Ap, -Bp*Cp
       zeros(2,10), Ap ];
Bt = [ -Bq*Dp, -Bq
       -Bq*Dp*Dp, -Bq*Dp
       -Bp*Dp, -Bp
       Bp, zeros(2,1) ];
Wt=diag([RMS2_d, RMS2_n]);
Xt=dlyap(At,Bt*Wt*Bt');

% Initialize optimization procedure and set options.
num_q0=num_q;                          % Initial guess.
opt=optimoptions('fmincon',...
                'Algorithm','sqp');     % SQP algorithm.
opt=optimoptions(opt,'Display','iter');  % Display results.
opt=optimoptions(opt,'TolX',1e-6);       % Termination tolerance for parameters.
opt=optimoptions(opt,'TolFun',1e-6);     % Termination tolerance for objective
                                        % function.
opt=optimoptions(opt,'TolCon',1e-6);     % Termination tolerance for constraints.
opt=optimoptions(opt,'GradObj','on');    % Use gradient.
opt=optimoptions(opt,'GradConstr','on'); % Use constraint Jacobian.
%
% For a first run of this m-file, you may want to uncomment the following
% two lines, which cause extra diagnostic information to be printed and an
% extra derivative check to be run.
%
%%opt=optimoptions(opt,'Diagnostics','on');
%%opt=optimoptions(opt,'DerivativeCheck','on','FinDiffType','central');

% Compute the trade-off curve.
dv_list=[20:-1:3];                     % List of desired values for M11.
results=zeros(length(dv_list),8);      % Initialization of results.
f_counter=0;                           % Counter for function evaluations.
i_counter=0;                           % Counter for iterations.
for i=1:length(dv_list)
  num_q0=num_q;                        % Initial guess.
  dv_M11=dv_list(i);
  [num_q,J,exitflag,output]=...
      fmincon(@(num_q) mcd_J(num_q,den_q,Cp,Dp,Xt,Wt,rho),num_q0,...
              [],[],[],[],[],[],...
              @(num_q) mcd_con(num_q,den_q,dv_M11),opt);
  f_counter=f_counter+output.funcCount;
  i_counter=i_counter+output.iterations;
  infn_M11=mcd_con(num_q,den_q,dv_M11)+dv_M11;
  results(i,:)=[dv_M11,J,infn_M11,num_q];

  disp(['Step ',num2str(i)])
  disp(['Results: ',num2str(results(i,1:2))])
  disp(' ')
end

disp(['Total number of function evaluations: ',num2str(f_counter)]);
disp(['Total number of iterations: ',num2str(i_counter)]);

plot(results(:,1),results(:,2));
xlabel('1/D_{min}');
ylabel('J');
```

mcd_J

The noise sensitivity J is calculated according to equation (14.16). The gradient of the noise sensitivity is calculated according to (14.17).

```
function [J,G]=mcd_J(num_q,den_q,Cp,Dp,Xt,Wt,rho)
% MCD_J  Calculates the function to be minimized (J) and its gradient (G)

% State space representations of Q
[Aq,Bq,Cq,Dq]=tf2ss(num_q,den_q);

% Compute noise sensitivity using Lyapunov theory
Ct=[ Cq, zeros(1,6), -Dq*Cp
       zeros(1,4), Cq, Dq*Cp, Cp-Dq*Dp*Cp ];
Dt=[ -Dq*Dp, -Dq
       Dp-Dq*Dp*Dp, -Dq*Dp ];

Zt=Ct*Xt*Ct'+Dt*Wt*Dt';
J=Zt(2,2)+rho*Zt(1,1);

if ( nargout > 1 )
   % Compute gradient of objective function.
   dc1=[[-den_q(2:5); eye(4)], zeros(5,6), [-Cp; zeros(4,2)]];
   dc2=[zeros(5,4), [-den_q(2:5); eye(4)], [Cp; zeros(4,2)], ...
         [-Dp*Cp; zeros(4,2)]];
   dd1=[[-Dp, -1]; zeros(4,2)];
   dd2=[[-Dp*Dp, -Dp]; zeros(4,2)];
   G=2*(dc2*Xt*Ct(2,:)'+dd2*Wt*Dt(2,:)'+...
          rho*(dc1*Xt*Ct(1,:)'+dd1*Wt*Dt(1,:)'));
end;
```

mcd_con

The ∞-norm of $M_{(1,1)}$ is calculated by setting up a fine grid of frequency points, evaluating the magnitude of the transfer function $M_{(1,1)}$ at each point, and taking the maximum of these magnitudes. The subgradient of $\|M_{(1,1)}\|_\infty$ is calculated according to (14.23).

```
function [c,ceq,Gc,Gceq]=mcd_con(num_q,den_q,dv_M11)
% MCD_CON  Calculates the constraints (c and ceq) and their Jacobians (GC
%          and GCeq)

ceq=[];   % No nonlinear equality constraints.
```

```
Gceq=[];

% Evaluate infinity norm of M11.
wr=[0:0.01:pi];
values_M11=freqz(num_q,den_q,wr);
[infn_M11,ind_M11]=max(abs(values_M11));

% Compute constraint for robustness to additive plant perturbation
c=infn_M11-dv_M11;

if ( nargout > 2 )
   % Compute subgradient of the infinity norm of M11
   w11=wr(ind_M11);
   buffer=[1; exp(-j*w11); exp(-j*2*w11); exp(-j*3*w11); exp(-j*4*w11)];
   Gc=real(infn_M11*buffer/(num_q*buffer));
end;
```

14.9 Discussion of the results

Figure 14.3 presents the trade-off curve between noise sensitivity and robustness to additive plant perturbations, calculated with the use of the MATLAB programs described in the previous section. This Figure leads to the important conclusion that decreasing $1/D_{\min}$ from 20 to 10 leads only to a moderate increase of the noise sensitivity. Hence, the controller can be made significantly more robust by allowing a slight increase in noise sensitivity.

As we have seen in Sections 14.6 and 14.7, it takes much effort to find an analytic expression for the gradient of the objective function and the subgradient of the constraint. Instead of calculating the gradients exactly, it is also possible to approximate them with finite differences. In general, however, it will take much more optimization steps to find the minimum. This is illustrated in table 14.1 for the computation of the trade-off curve. This table shows a significant increase in the number of function evaluations when the gradients are approximated.

As we have already mentioned before, the computation of the trade-off curve boils down to solving a number of convex optimization problems. However, we did not use a convex optimization method to solve it. One reason for this is that the *MATLAB Optimization Toolbox* does not contain convex optimization routines. A second, more important reason is that for the particular (low dimensional) problem considered in this chapter, the SQP algorithm converges faster than a convex optimization algorithm does. Recall that convex optimization routines start with a particular region in which the optimum can be found, and in each optimization step try to decrease this region. The SQP algorithm starts with an initial estimate of the optimum and updates it in every optimization step. For the particular problem at hand, we have some very good initial estimates, which are close to the optimum. For the first point of the trade-off curve, we use the LQG controller as an initial guess, and for the next points we use the previous point as initial guess. We can, however, not use these initial estimates in the convex optimization routines because they are not feasible. Furthermore, the noise sensitivity is not only a convex function, but also quadratic with respect to the parameters. Therefore, we may conclude that, for this particular problem, the SQP algorithm starting from an initial estimate close to the optimum can converge much faster than a convex optimization algorithm starting from a relatively large region around the optimum.

From the previous paragraph, we may not conclude that the SQP algorithm is always better than a convex optimization routine. Often, the opposite is true. When the problem is convex, but not

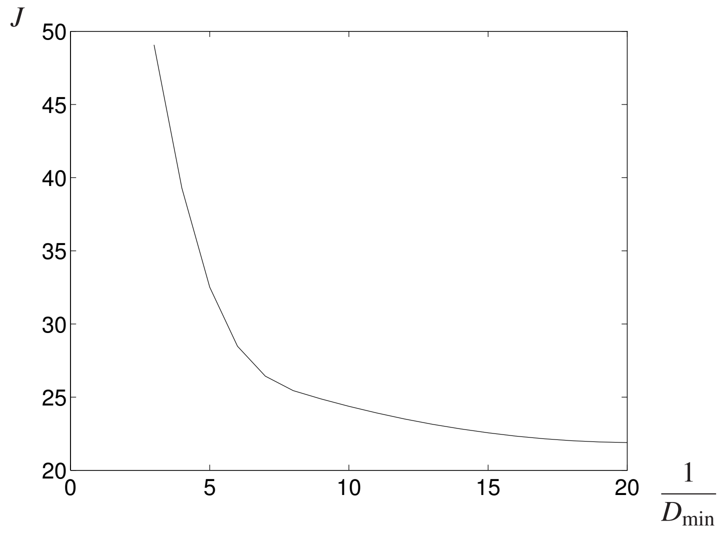


Figure 14.3: Trade-off between noise sensitivity J and robustness for plant perturbations $D_{\min}$.

quadratic or not even smooth, the SQP algorithm converges much slower and might not even find the real optimum. Therefore, when SQP is used, it is recommended to check the Kuhn-Tucker conditions for the solution found. Using a convex algorithm guarantees that the solution found equals the optimum.

We could ask ourselves the following question: if the convex optimization routines perform worse, why do we need to formulate the problem as a convex problem? Note that if we do not formulate the optimization problem as a convex problem, then the complexity of the problem would increase. Moreover, if the problem is formulated as a convex optimization problem we have the guarantee to find the global optimum, because there are no other local minima. To illustrate the increase in complexity, say that we allow the poles of Q to change. Then the system matrix A_q is no longer independent of the parameters we want to optimize. Therefore, the expression for the gradient of the noise sensitivity will become more complex, since the solution of the Lyapunov equation (14.16) will depend on the poles of Q. Even worse, the robustness constraint is no longer convex, and as a consequence a subgradient cannot be found using the method presented in Section 14.7.

In practice, we are of course not only interested in the effect the zeros of Q have on the trade-off curve, but also in the effect the poles of Q have. A common way to deal with this problem is to take a truncated impulse response sequence for Q, that is

$$Q(q) = \sum_{i=0}^{N} \theta_i q^{-i} \tag{14.24}$$

for N sufficiently large. It is easy to verify that for this choice of Q the optimization problem stays convex. Since N is large, the optimization space is of a very high dimension. In this case, it is expected that a convex optimization algorithm outperforms the SQP algorithm. Due to the large value of N, the resulting controller will have a very high order, which is not desirable. A common way to deal with this problem is: first, use the impulse sequence (14.24) to calculate a controller via convex optimization; then, use model reduction techniques to approximate this impulse sequence by a ratio of two low-order polynomials in z.

	Number of function evaluations	Number of gradient evaluations
Exact calculation	887	222
Approximation	2763	293

Table 14.1: The number of function and gradient evaluations needed for the determination of the trade-off curve, when the gradients are calculated using an analytic expression and when they are approximated by finite differences.

Exercises

Exercise 14.1 Determine the transfer function G of the generalized plant for the basic feedback loop in Figure 12.9 and the signals w, u, z, v stated in (12.4).

Part III

Appendices

Appendix A

Basic State Space Operations

In this appendix we show how state space descriptions change under basic operations such as cascade connection, parallel connection, change of variables, etc.

Consider the following discrete-time state space model for a linear time-invariant system:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \quad . \end{aligned}$$

We use the following notation for the transfer function G of this system:

$$G(q) = C(qI-A)^{-1}B + D = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \quad .$$

Note that the entity at the right-hand side of this equations is *not* a matrix but just a short-hand notation for $C(qI-A)^{-1}B+D$!

A.1 Cascade connection or series connection

$$G(q) \quad \longrightarrow \quad \hat{G}(q) = G_1(q) \cdot G_2(q)$$

$$G_1(q) \;=\; \left[\begin{array}{c|c} A_1 & B_1 \\ \hline C_1 & D_1 \end{array}\right] \qquad\qquad G_2(q) = \left[\begin{array}{c|c} A_2 & B_2 \\ \hline C_2 & D_2 \end{array}\right]$$

$$\begin{aligned} \hat{G}(q) &= \left[\begin{array}{c|c} A_1 & B_1 \\ \hline C_1 & D_1 \end{array}\right] \cdot \left[\begin{array}{c|c} A_2 & B_2 \\ \hline C_2 & D_2 \end{array}\right] \\ &= \left[\begin{array}{cc|c} A_1 & B_1C_2 & B_1D_2 \\ 0 & A_2 & B_2 \\ \hline C_1 & D_1C_2 & D_1D_2 \end{array}\right] \\ &= \left[\begin{array}{cc|c} A_2 & 0 & B_2 \\ B_1C_2 & A_1 & B_1D_2 \\ \hline D_1C_2 & C_1 & D_1D_2 \end{array}\right] \end{aligned}$$

A.2 Parallel connection

$$G(q) \quad \longrightarrow \quad \hat{G}(q) = G_1(q) + G_2(q)$$

$$G_1(q) = \left[\begin{array}{c|c} A_1 & B_1 \\ \hline C_1 & D_1 \end{array}\right] \qquad G_2(q) = \left[\begin{array}{c|c} A_2 & B_2 \\ \hline C_2 & D_2 \end{array}\right]$$

$$\begin{aligned} \hat{G}(q) &= \left[\begin{array}{c|c} A_1 & B_1 \\ \hline C_1 & D_1 \end{array}\right] + \left[\begin{array}{c|c} A_2 & B_2 \\ \hline C_2 & D_2 \end{array}\right] \\ &= \left[\begin{array}{cc|c} A_1 & 0 & B_1 \\ 0 & A_2 & B_2 \\ \hline C_1 & C_2 & D_1 + D_2 \end{array}\right] \end{aligned}$$

A.3 Change of variables

$$\begin{array}{lll} x \longrightarrow \hat{x} = Tx & & T \text{ is invertible} \\ u \longrightarrow \hat{u} = Pu & & P \text{ is invertible} \\ y \longrightarrow \hat{y} = Ry & & \end{array}$$

$$G(q) = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

$$\hat{G}(q) = \left[\begin{array}{c|c} \hat{A} & \hat{B} \\ \hline \hat{C} & \hat{D} \end{array}\right] = \left[\begin{array}{c|c} TAT^{-1} & TBP^{-1} \\ \hline RCT^{-1} & RDP^{-1} \end{array}\right]$$

A.4 State feedback

$$u \quad \longrightarrow \quad \hat{u} + Fx$$

$$G(q) = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

$$\hat{G}(q) = \left[\begin{array}{c|c} \hat{A} & \hat{B} \\ \hline \hat{C} & \hat{D} \end{array}\right] = \left[\begin{array}{c|c} A + BF & B \\ \hline C + DF & D \end{array}\right]$$

A.5 Output injection

$$x(k+1) = Ax(k) + Bu(k) \quad \longrightarrow \quad \hat{x}(k+1) = A\hat{x}(k) + Bu(k) + Hy(k)$$

$$G(q) = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

$$\hat{G}(q) = \left[\begin{array}{c|c} \hat{A} & \hat{B} \\ \hline \hat{C} & \hat{D} \end{array}\right] = \left[\begin{array}{c|c} A+HC & B+HD \\ \hline C & D \end{array}\right]$$

A.6 Transpose

$$G(q) \quad \longrightarrow \quad \hat{G}(q) = G^T(q)$$

$$G(q) = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

$$\hat{G}(q) = \left[\begin{array}{c|c} A^T & C^T \\ \hline B^T & D^T \end{array}\right]$$

A.7 Left (right) inversion

$$G(q) \quad \longrightarrow \quad \hat{G}(q) = G^+(q)$$

G^+ is a left (right) inverse of G.

$$G(q) = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

$$\hat{G}(q) = \left[\begin{array}{c|c} A - BD^+C & BD^+ \\ \hline -D^+C & D^+ \end{array}\right]$$

where D^+ is the left (right) inverse of D.
For $G(q)$ square, we obtain $G^+(q) = G^{-1}(q)$ and $D^+ = D^{-1}$.

Appendix B

Jury's Stability Criterion

In this appendix we present Jury's stability criterion. Consider a linear time-invariant (LTI) plant $\mathscr{G}$ with transfer function $G(q)$ where q^{-1} is the unit delay operator (i.e., $q^{-1}u(k) = u(k-1)$). Let $G(z)$ be the z-transform[1] of the impulse response of the plant $\mathscr{G}$. Now we write $G(z)$ as

$$G(z) = \frac{G_{\text{num}}(z)}{G_{\text{den}}(z)}$$

with G_{num} and G_{den} respectively the numerator and denominator of G. Furthermore, assume that

$$G_{\text{den}}(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_2 z^2 + a_1 z + a_0 \tag{B.1}$$

with all coefficients a_i real, and with $a_n > 0$.

Now a *necessary and sufficient* condition for the plant $\mathscr{G}$ to be stable is that all roots of $G_{\text{den}}(z)$ lie inside the unit circle. As computing the roots of $G_{\text{den}}(z)$ requires numerical approaches for n larger than 4, it is sometimes useful to have necessary conditions for stability that can be checked by inspection. One set of such conditions can be derived from Jury's stability criterion [21] and can be formulated as follows.

Proposition B.1 *Let $\mathscr{G}$ be an LTI plant with transfer function $G = \dfrac{G_{\text{num}}}{G_{\text{den}}}$ and such that G_{den} can be written as (B.1) with all coefficients a_i real and with $a_n > 0$. If $\mathscr{G}$ is stable, then*

$$G_{\text{den}}(1) > 0 \tag{B.2}$$

$$G_{\text{den}}(-1) > 0 \qquad \textit{if } n \textit{ is an even integer} \tag{B.3}$$

$$G_{\text{den}}(-1) < 0 \qquad \textit{if } n \textit{ is an odd integer} \tag{B.4}$$

$$|a_0| < a_n \ . \tag{B.5}$$

In general the conditions (B.2)–(B.5) are necessary (but not sufficient) conditions for the plant $\mathscr{G}$ to be stable. However, for a second-order system, the conditions (B.2)–(B.5) are *necessary and sufficient*. Note that these conditions are also necessary and sufficient for first-order systems.

Example B.2 Consider the plant $\mathscr{G}$ with transfer function

$$G(q) = \frac{5 + 2q^{-1}}{3 + 2q^{-1} + q^{-2}} = \frac{5q^2 + 2q}{3q^2 + 2q + 1} \ .$$

[1] Loosely speaking this z-transform can be obtained by replacing the q-operator in the transfer function $G(q)$ by the z-variable.

Let us first check whether $\mathscr{G}$ is stable by computing the roots of $G_{\text{den}}(z) = 3z^2 + 2z + 1$ and verifying whether they lie inside the unit circle. The roots $z_{1,2}$ of G_{den} are given by $z_{1,2} \approx -0.3333 \pm 0.4714j$ with $|z_{1,2}| \approx 0.5774$. So the plant $\mathscr{G}$ is stable. Now we verify this again by applying the conditions (B.2)–(B.5). We have

$$\begin{aligned} &G_{\text{den}}(1) = 6 > 0 \\ &G_{\text{den}}(-1) = 2 > 0 \qquad (n = 2) \\ &|a_0| = 1 < 3 = a_2 \ . \end{aligned}$$

So the necessary and sufficient conditions for stability indeed hold. □

Appendix C

Singular Value Decomposition

If $A \in \mathbb{R}^{m\times n}$ then there exist orthogonal matrices

$$U = \begin{bmatrix} u_1 & \cdots & u_m \end{bmatrix} \in \mathbb{R}^{m\times m}$$

and

$$V = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} \in \mathbb{R}^{n\times n}$$

such that

$$U^T A V = \Sigma$$

where

$$\Sigma = \left[\begin{array}{ccc|ccc} \sigma_1 & & & 0 & \cdots & 0 \\ & \ddots & & \vdots & & \vdots \\ & & \sigma_m & 0 & \cdots & 0 \end{array}\right] \quad \text{for } m \leqslant n$$

$$\Sigma = \left[\begin{array}{ccc} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ \hline 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{array}\right] \quad \text{for } m \geqslant n$$

where $\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_p \geqslant 0$, $p = \min(m,n)$. This means that

$$A = U\Sigma V^T$$

This decomposition of A is known as *the singular value decomposition of* (SVD) A. The σ_i are known as the *singular values* of A (and are usually arranged in descending order).

We have [19, 22]:

Property C.1 *Let $U\Sigma V^T$ be the SVD of A. U contains the eigenvectors of AA^T. V contains the eigenvectors of A^TA. The left-most or top-most block of Σ is a diagonal matrix whose entries are the nonnegative square roots of the eigenvalues A^TA (for $m \geqslant n$) or AA^T (for $m \leqslant n$).*

Property C.2 *Let $\sigma_{\min}$ and $\sigma_{\max}$ be respectively the smallest and the largest singular value of a square matrix A. Let λ_i by the ith eigenvalue of A. Then we have*

$$\sigma_{\min} \leqslant |\lambda_i| \leqslant \sigma_{\max} \qquad \textit{for all } i\,.$$

Appendix D

Least Squares Problems

An important class of optimization problems are least squares problems. The least squares criterion is the most widely used criterion in signal processing, communications and control. For a certain class of least squares problems, we can obtain a closed-form expression of the solution. This particular class of least squares solutions can often be used to generate initial guesses for more complicated, nonlinear or constrained optimization problems. Chapter 14 gives an example of this; there we compute an LQG controller as an initial guess for optimizing a controller subject to a robustness constraint.

In this chapter we take a look at two least squares problems which have an analytic solution, namely the ordinary least squares problem and the total least squares problem. We also discuss a recently proposed modification of the total least squares problem, the robust least squares problem.

D.1 Ordinary least squares

Given the data matrix $A \in \mathbb{R}^{m\times n}$ and the observation vector $b \in \mathbb{R}^m$ with $m \geq n$. Consider the problem of finding a vector x such that $Ax \approx b$. It is assumed that the matrix A has full rank. When $m > n$ the system $Ax \approx b$ is often called an overdetermined system of equations. This system of equations will only have an exact solution if b is in the column space of A. This suggests the following minimization problem

$$\min_x \|Ax - b\|_p$$

where p is a suitable vector norm. The most widely used norm in this context is the 2-norm (Euclidean norm). The 2-norm of the n-dimensional vector x is defined by

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{D.1}$$

The 2-norm is popular because of two reasons:

- The objective function $f(x) = \|Ax - b\|_2^2$ is a differentiable function.
- The 2-norm is preserved under orthogonal transformation. This is an interesting fact from an algorithmic point of view, because it implies that we can seek an orthogonal matrix Q such that the equivalent problem of minimizing $\|Q^T Ax - Q^T b\|_2$ is “easy” to solve [19].

Use of the 2-norm in equation (D.1) gives rise to the *ordinary least squares problem*: find a vector $x \in \mathbb{R}^n$ which satisfies

$$\min_x \|Ax - b\|_2^2 \tag{D.2}$$

The objective function of this minimization problem can be written as

$$\begin{aligned} f(x) &= \|Ax-b\|_2^2 \\ &= (Ax-b)^T(Ax-b) \\ &= x^TA^TAx - x^TA^Tb - b^TAx + b^Tb \end{aligned}$$

From this expression it is easy to compute the gradient

$$\nabla f(x) = 2A^TAx - 2A^Tb$$

The solution to the least squares problem is found by setting $\nabla f(x) = 0$. It follows that the solution must satisfy the *normal equations*, which are given by

$$A^TAx = A^Tb$$

Now we can formulate the following theorem.

Theorem D.1 *If rank$(A) = n$, the unique solution of the least squares problem (D.2) is given by*

$$x = (A^TA)^{-1}A^Tb$$

The solution for the rank deficient case is described in [19].

An important application of least squares problems is parameter estimation where the vector b is noisy. In this case, the parameters x are related to the noisy observations b by

$$Ax = b_0, \qquad b = b_0 + \delta b$$

where δb is a random error. Due to this error the vector b is not in the column space of A. The least squares solution is obtained by projection of the vector b onto this column space.

Recall that the column space of a matrix $C \in \mathbb{R}^{m\times n}$ is defined as

$$\mathscr{R}(C) = \{y \in \mathbb{R}^m : y = Cx \text{ for some } x \in \mathbb{R}^n\}$$

Another equivalent formulation of the least squares problem is the following: given an overdetermined system of m linear equations $Ax \approx b$ in n unknowns x, with $m \geq n$. The least squares problem is equivalent to solving the following minimization problem:

$$\min_{\hat{b}} \|b - \hat{b}\|_2 \quad \text{subject to: } \hat{b} \in \mathscr{R}(A)$$

where $\mathscr{R}(A)$ denotes the column space of the matrix A. Any x satisfying

$$Ax = \hat{b}$$

is called a least squares solution.

D.2 Total least squares

This section is based on Chapter 2 of the book on total least squares problems by Van Huffel and Vandewalle [44].

The underlying assumption of the least squares problem is that errors only occur in the vector b and that the matrix A is exactly known. The *total least squares problem* also takes into account errors in the matrix A. This problem is also known as the *errors in variables problem*.

The total least squares problem can be formulated as follows: Given an overdetermined system of m linear equations $Ax \approx b$ in n unknowns x, with $m \geq n$. The total least squares problem is equivalent to solving the following minimization problem:

$$\min_{[\hat{A}\ \hat{b}]} \left\| [A\ b] - [\hat{A}\ \hat{b}] \right\|_{\mathrm{F}} \qquad \text{subject to: } \hat{b} \in \mathscr{R}(\hat{A}) \tag{D.3}$$

where $\mathscr{R}(\hat{A})$ denotes the column space of the matrix $\hat{A}$, and $\|\cdot\|_{\mathrm{F}}$ denotes the *Frobenius norm*; the Frobenius norm of a matrix $C \in \mathbb{R}^{m\times n}$ is defined by

$$\|C\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}^2}$$

Any x satisfying

$$\hat{A}x = \hat{b}$$

is called a total least squares solution.

Observe the resemblance between the total least squares problem and the alternative formulation of the ordinary least squares problem at the end of the previous section.

Before we are going to look at the solution of the total least squares problem, we review an important result from linear algebra. The singular value decomposition (SVD) of a matrix $C \in \mathbb{R}^{m\times n}$ with $\mathrm{rank}(C) = r < \min(m,n)$ is given by

$$C = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \tag{D.4}$$

where $U_1 \in \mathbb{R}^{m\times r}$, $U_2 \in \mathbb{R}^{m\times(m-r)}$, $V_1 \in \mathbb{R}^{n\times r}$, $V_2 \in \mathbb{R}^{n\times(n-r)}$ and $\Sigma = \mathrm{diag}(\sigma_1,\ldots,\sigma_n)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$. Recall that the null space of a matrix $C \in \mathbb{R}^{m\times n}$ is defined as

$$\mathscr{N}(C) = \{x \in \mathbb{R}^n : Cx = 0\}$$

From equation (D.4) we have the following two important relations

$$\begin{aligned} \mathscr{R}(C) &= \mathscr{R}(U_1) \\ \mathscr{N}(C) &= \mathscr{R}(V_2) \end{aligned}$$

To solve the total least square problem, we need the Eckart-Young matrix approximation theorem, which is stated below.

Theorem D.2 *Let the SVD of $C \in \mathbb{R}^{m\times n}$ be given by $C = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T$ with $r = \mathrm{rank}(C)$. If $k < r$ and $C_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$ then*

$$\left\{\min_{D} \|C - D\|_{\mathrm{F}} : \mathrm{rank}(D) = k\right\} = \|C - C_k\|_{\mathrm{F}} = \sqrt{\sum_{i=k+1}^{p} \sigma_i^2}, \;\; p = \min(m,n)$$

The proof of this theorem can be found in [19].

The following theorem provides conditions for the uniqueness and existence of a total least squares solution. It also provides a closed-form expression for the solution.

Theorem D.3 *Suppose that the SVD of the full rank matrix $A \in \mathbb{R}^{m\times n}$ is given by*

$$A = U'\Sigma'(V')^T, \qquad \Sigma' = diag(\sigma_1', \dots, \sigma_n') \tag{D.5}$$

and suppose that the SVD of the matrix $[A\ b] \in \mathbb{R}^{m\times(n+1)}$ is given by

$$[A\ b] = U\Sigma V^T, \qquad \Sigma = diag(\sigma_1, \dots, \sigma_{n+1})$$

If $\sigma_n' > \sigma_{n+1}$ then

$$[\hat{A}\ \hat{b}] = U\hat{\Sigma}V^T, \qquad \hat{\Sigma} = diag(\sigma_1, \dots, \sigma_n, 0) \tag{D.6}$$

solves the total least squares problem (D.3) and

$$x = (A^TA - \sigma_{n+1}^2 I)^{-1}A^Tb \tag{D.7}$$

exists and is the unique solution to $\hat{A}x = \hat{b}$.

Proof: If $\sigma_{n+1} \neq 0$, $[A\ b]$ is of rank $n+1$. Thus, the null space of $[A\ b]$ contains no nonzero vectors, and the equation $Ax = b$ has no solution. To obtain a solution the rank of $[A\ b]$ must be reduced to n. According to the Eckart-Young theorem the best rank n approximation of $[A\ b]$ is given by equation (D.6), and the minimal correction is

$$\left\{ \min_{[\hat{A}\ \hat{b}]} \left\| [A\ b] - [\hat{A}\ \hat{b}] \right\|_{\mathrm{F}} : \mathrm{rank}([\hat{A}\ \hat{b}]) = n \right\} = \sigma_{n+1}$$

Now we have the compatible system of equations

$$[\hat{A}\ \hat{b}] \begin{bmatrix} x \\ -1 \end{bmatrix} = 0$$

The solutions to this system of equations are the vectors αv_{n+1}, $\alpha \in \mathbb{R}$ (v_{n+1} is the last column of V) belonging to $\mathcal{N}([\hat{A}\ \hat{b}])$. The total least squares solution x is obtained by scaling v_{n+1} until its last component is -1, that is

$$\begin{bmatrix} x \\ -1 \end{bmatrix} = \frac{-1}{v_{n+1,n+1}} v_{n+1} \tag{D.8}$$

Since $\mathcal{N}([\hat{A}\ \hat{b}])$ has dimension one, the solution must be unique.

If $\sigma_{n+1} = 0$, $[A\ b]$ is of rank n, and the system $Ax \approx b$ is compatible. Thus, no approximation is needed. The exact solution is given by (D.8). Obviously, this solution is unique.

Since the singular vectors v_i are eigenvectors of $[A\ b]^T[A\ b]$, x always satisfies the following eigenvector equation

$$[A\ b]^T[A\ b] \begin{bmatrix} x \\ -1 \end{bmatrix} = \begin{bmatrix} A^TA & A^Tb \\ b^TA & b^Tb \end{bmatrix} \begin{bmatrix} x \\ -1 \end{bmatrix} = \sigma_{n+1}^2 \begin{bmatrix} x \\ -1 \end{bmatrix}$$

Equation (D.7) follows directly from the top part of this expression.

We have to prove that if $\sigma'_n > \sigma_{n+1}$ then there exists a solution x, or equivalently there exists a vector v_{n+1} whose $(n+1)$th element is nonzero. We will do this by contradiction. Suppose that for $y \neq 0$ we have

$$[A\ b]^T [A\ b] \begin{bmatrix} y \\ 0 \end{bmatrix} = \sigma_{n+1}^2 \begin{bmatrix} y \\ 0 \end{bmatrix}$$

Clearly, it follows that $A^T A y = \sigma_{n+1}^2 y$. From equation (D.5), we know that the smallest eigenvalue of $A^T A$ is $(\sigma'_n)^2$. Hence, it cannot be that $\sigma_{n+1} < \sigma'_n$. □

An important application of total least squares problems is parameter estimation where both the vector b and the matrix A are noisy. In this case, the parameters x are related to the noisy observations by

$$A_0 x = b_0, \qquad A = A_0 + \delta A \text{ and } b = b_0 + \delta b$$

where δb and δA are random errors.

D.3 Robust least squares

The results stated in this section were taken from a recent article by Chandrasekaran *et al.* [5]. Another approach to robust least squares, which is based on second-order cone programming has been described by El Ghaoui *et al.* [15].

As shown in the previous section, the total least squares solution takes into account errors on the data matrix A. In some practical situations, it still exhibits certain drawbacks that degrade its performance. It may unnecessarily over-emphasize the effect of noise and can, therefore, lead to overly conservative results. Consider, for example, a matrix A which is almost exactly known, that is, the errors on A are very small. If the vector b is far from the column space of A, the total least solution rotates $[A\ b]$ and thus overly correct the matrix A.

Chandrasekaran *et al.* [5] introduced the *robust least squares problem* which does not suffer from the drawbacks mentioned above. The robust least squares problem takes into account some prior bounds on the size of the allowable perturbations. This guarantees that the effect of the uncertainties will never be unnecessarily over-estimated.

The robust least squares problem can be formulated as follows: given an overdetermined system of m linear equations $Ax \approx b$ in n unknowns x, with $m \geq n$. Assume that the "true" matrix is $A + \delta A$ and that we have an upper bound on the perturbation δA: $\|\delta A\|_2 \leq \eta$, $\eta > 0$. Assume that the "true" vector is $b + \delta b$ and that we have an upper bound on the perturbation δb: $\|\delta b\|_2 \leq \eta_b$, $\eta_b \geq 0$. The robust least squares problem is equivalent to solving the following constrained min-max problem:

$$\min_x \max \left\{ \|(A + \delta A)x - (b + \delta b)\|_2 : \|\delta A\|_2 \leq \eta, \|\delta b\|_2 \leq \eta_b \right\} \tag{D.9}$$

The robust least squares approach seeks a solution x satisfying $Ax \approx b$ that performs "best" in the worst-possible scenario. That is, it minimizes the worst-possible residual over the class of perturbations δA and δb.

D.3.1 Reducing the min-max problem to a minimization problem

The constrained min-max problem (D.9) is equivalent to

$$\min_x \left(\|Ax - b\|_2 + \eta \|x\|_2 + \eta_b \right) \tag{D.10}$$

In other words, we have

$$\max\Big\{ \|(A+\delta A)x-(b+\delta b)\|_2 : \|\delta A\|_2 \le \eta, \|\delta b\|_2 \le \eta_b \Big\} = \|Ax-b\|_2 + \eta\,\|x\|_2 + \eta_b$$

This can be seen from the following upper bound

$$\begin{aligned} \|(A+\delta A)x-(b+\delta b)\|_2 &\le \|Ax-b\|_2 + \|\delta A\|_2\,\|x\|_2 + \|\delta b\|_2 && \text{(D.11)}\\ &\le \|Ax-b\|_2 + \eta\,\|x\|_2 + \eta_b && \text{(D.12)} \end{aligned}$$

That there exist a δA and a δb for which this bound is achievable, is left as an exercise for the reader (see Exercise D.1).

D.3.2 Solving the minimization problem

To solve (D.10), we define the objective function

$$f(x) = \|Ax-b\|_2 + \eta\,\|x\|_2 + \eta_b$$

This is a convex function and therefore, any local minimum of $f(x)$ is also a global minimum. At any local minimum it must hold that the gradient of $f(x)$ equals zero, or $f(x)$ is not differentiable. Note that $f(x)$ is not differentiable only at $x = 0$ and at any x satisfying $Ax - b = 0$.

We first consider the case in which $f(x)$ is differentiable. The gradient of $f(x)$ is given by

$$\begin{aligned} \nabla f(x) &= \frac{1}{\|Ax-b\|_2} A^T(Ax-b) + \frac{\eta}{\|x\|_2} x \\ &= \frac{1}{\|Ax-b\|_2}\Big((A^TA+\alpha I)x - A^Tb \Big) \end{aligned}$$

where we have introduced the positive real number

$$\alpha = \frac{\eta\,\|Ax-b\|_2}{\|x\|_2} \tag{D.13}$$

The solution to (D.10) is found by setting $\nabla f(x) = 0$. We get

$$x = (A^TA+\alpha I)^{-1}A^Tb \tag{D.14}$$

D.3.3 The secular equation

We still have to determine the parameter α that corresponds to x, in equation (D.14). Since the parameter α is defined by (D.13), we are going to derive expressions for $\|x\|_2$ and $\|Ax-b\|_2$. Assume that A has full rank, then its SVD is given by

$$A = U \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^T$$

We partition the vector U^Tb into

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = U^Tb$$

where $b_1 \in \mathbb{R}^n$ and $b_2 \in \mathbb{R}^{m-n}$.

Now we can rewrite equation (D.14) as

$$x = V(\Sigma^2 + \alpha I)^{-1}\Sigma b_1$$

Hence,

$$\|x\|_2 = \|\Sigma(\Sigma^2 + \alpha I)^{-1} b_1\|_2$$

We also have

$$\begin{aligned} b - Ax &= b - AV(\Sigma^2 + \alpha I)^{-1}\Sigma b_1 \\ &= U\left(U^T b - \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} (\Sigma^2 + \alpha I)^{-1}\Sigma b_1\right) \\ &= U\begin{bmatrix} b_1 - \Sigma^2(\Sigma^2 + \alpha I)^{-1} b_1 \\ b_2 \end{bmatrix} \\ &= U\begin{bmatrix} \alpha(\Sigma^2 + \alpha I)^{-1} b_1 \\ b_2 \end{bmatrix} \end{aligned}$$

and thus,

$$\|b - Ax\|_2^2 = \|b_2\|_2^2 + \alpha^2\|(\Sigma^2 + \alpha I)^{-1} b_1\|_2^2$$

Therefore, equation (D.13) for α reduces to the following nonlinear equation that is only a function of α and the given data: A, b, and η.

$$\alpha = \frac{\eta\sqrt{\|b_2\|_2^2 + \alpha^2\|(\Sigma^2 + \alpha I)^{-1} b_1\|_2^2}}{\|\Sigma(\Sigma^2 + \alpha I)^{-1} b_1\|_2}$$

It is clear that α is a positive solution of the previous equation if and only if it is a positive root of

$$\mathscr{G}(\alpha) = b_1^T(\Sigma^2 - \eta^2 I)(\Sigma^2 + \alpha I)^{-2} b_1 - \frac{\eta^2}{\alpha^2}\|b_2\|_2^2 \tag{D.15}$$

The equation $\mathscr{G}(\alpha) = 0$ is called a *secular equation* [19].

D.3.4 The solution to the robust least squares problem

In [5] the following lemma has been proved.

Lemma D.4 *Assume that b does not belong to the column space of A. Then, the function $\mathscr{G}(\alpha)$ in (D.15) has a unique positive root if and only if*

$$\eta < \frac{\|A^T b\|_2}{\|b\|_2} \tag{D.16}$$

It can be concluded that, if b does not belong to the column space of A, and the condition (D.16) holds, the global minimum of $f(x)$ is obtained for x given by (D.14) with α the unique positive root of $\mathscr{G}(\alpha)$. If the condition (D.16) does not hold, $\mathscr{G}(\alpha)$ has no positive root. Then, the global minimum is obtained for $x = 0$, because under the assumption that b does not belong to the column space of A, the function $f(x)$ is not differentiable for $x = 0$.

These observations allow us to formulate the following theorem, concerning the solution of the robust least squares problem.

Theorem D.5 *Consider the robust least squares problem (D.9) Suppose that A has full rank, and that b does not belong to the column space of A. Let the SVD of A be given by*

$$A = U \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^T$$

Partition the vector $U^T b$ into

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = U^T b$$

where $b_1 \in \mathbb{R}^n$ and $b_2 \in \mathbb{R}^{(m-n)}$. Define

$$\tau = \frac{\|A^T b\|_2}{\|b\|_2}$$

and

$$\mathscr{G}(\alpha) = b_1^T(\Sigma^2 - \eta^2 I)(\Sigma^2 + \alpha I)^{-2} b_1 - \frac{\eta^2}{\alpha^2}\|b_2\|_2^2$$

- *If $\eta \geq \tau$, the unique solution to the robust least squares problem is $x = 0$.*
- *If $\eta < \tau$, the unique solution to the robust least squares problem is*

  $$x = (A^T A + \alpha I)^{-1} A^T b$$

  where α is the unique positive root of the secular equation $\mathscr{G}(\alpha) = 0$.

Note that the root of the secular equation can be found using numerical techniques, such as the Gauss-Newton method.

The robust least squares solution for the case when b does belong to the column space of A can be found in [5].

Exercises

Exercise D.1 Show that there exist a δA and a δb for which the upper bound of (D.12) is achievable.

Bibliography

[1] K.J. Aström and B. Wittenmark, *Computer Controlled Systems: Theory and Applications*. Englewood Cliffs, New Jersey: Prentice Hall, 1984.

[2] P.T. Boggs and J.W. Tolle, "Sequential quadratic programming," *Acta Numerica*, vol. 4, pp. 1–51, 1995.

[3] S. Boyd and C. Barratt, *Linear controller design, limits of performance*. Englewood Cliffs, New Jersey: Prentice Hall, 1991.

[4] S. Boyd, C. Barratt, and S. Norman, "Linear controller design: Limits of performance via convex optimization," *Proceedings of the IEEE*, vol. 78, no. 3, pp. 529–574, Mar. 1990.

[5] S. Chandrasekaran, G.H. Golub, M. Gu, and A.H. Sayed, "Parameter estimation in the presence of bounded data uncertainties," *SIAM Journal on Matrix Analysis and Applications*, vol. 19, no. 1, pp. 235–252, 1998.

[6] S. Chen and S.A. Billings, "Neural networks for nonlinear dynamic system modelling and identification," *International Journal of Control*, vol. 56, no. 2, pp. 319–346, 1992.

[7] T.F. Coleman and Y. Li, "On the convergence of reflective Newton methods for large-scale nonlinear minimization subject to bounds," *Mathematical Programming*, vol. 67, no. 2, pp. 189–224, 1994.

[8] T.F. Coleman and Y. Li, "An interior, trust region approach for nonlinear minimization subject to bounds," *SIAM Journal on Optimization*, vol. 6, pp. 418–445, 1996.

[9] T.F. Coleman and Y. Li, "A reflective Newton method for minimizing a quadratic function subject to bounds on some of the variables," *SIAM Journal on Optimization*, vol. 6, no. 4, pp. 1040–1058, 1996.

[10] G.B. Dantzig and M.N. Thapa, *Linear Programming 1: Introduction*. Springer Series in Operations Research and Financial Engineering, New York, New York: Springer-Verlag, 1997.

[11] L. Davis, ed., *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.

[12] V.F. Demyanov and L.V. Vasilev, *Nondifferentiable Optimization, Optimization Software*. Springer-Verlag, 1985.

[13] J.C. Doyle, B.A. Francis, and A.R. Tannenbaum, *Feedback Control Theory*. New York: Macmillan Publishing Company, 1992.

[14] R.W. Eglese, "Simulated annealing: A tool for operations research," *European Journal of Operational Research*, vol. 46, pp. 271–281, 1990.

[15] L. El Ghaoui and H. Lebret, "Robust solutions to least-squares problems with uncertain data," *SIAM Journal on Matrix Analysis and Appplications*, vol. 18, no. 4, Oct. 1997.

[16] R. Fletcher, *Practical Methods of Optimization, Volume 1: Unconstrained Optimization*. Chichester, UK: John Wiley & Sons, 1980.

[17] F. Glover and M. Laguna, *Tabu Search*. Boston: Kluwer Academic Publishers, 1997.

[18] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, Massachusetts: Addison-Wesley, 1989.

[19] G.H. Golub and C.F. Van Loan, *Matrix Computations*. Baltimore, Maryland: The John Hopkins University Press, 3rd ed., 1996.

[20] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*. Berlin, Germany: Springer-Verlag, 1988.

[21] E.I. Jury, *Theory and Application of the z-Transform Method*. New York: John Wiley & Sons, 1964.

[22] T. Kailath, *Linear Systems*. Englewood Cliffs, New Jersey: Prentice-Hall, 1980.

[23] R.M. Karp, "An introduction to randomized algorithms," *Discrete Mathematics*, vol. 34, pp. 165–201, 1991.

[24] H. Kwakernaak and R. Sivan, *Modern Signals and Systems*. Englewood Cliffs, New Jersey: Prentice Hall, 1991.

[25] C. Lawson and R. Hanson, *Solving Least Squares Problems*, vol. 15 of *Classics in Applied Mathematics*. Philadelphia, Pennsylvania: SIAM, 1995.

[26] C.E. Lemke, "On complementary pivot theory," in *Mathematics of the Decision Sciences, Part 1* (G.B. Dantzig and A.F. Veinott, eds.), pp. 95–114, Providence, Rhode Island: American Mathematical Society, 1968.

[27] L. Ljung and T. Glad, *Modeling of Dynamic Systems*. Englewood Cliffs, New Jersey: Prentice Hall, 1994.

[28] K.S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, pp. 4–27, 1990.

[29] S.G. Nash and A. Sofer, *Linear and Nonlinear Programming*. New York: McGraw-Hill, 1996.

[30] J.A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, pp. 308–313, 1965.

[31] A.S. Nemirovsky and D.B. Yudin, *Problem Complexity and Method Efficiency in Optimization*. Chichester: Wiley-Interscience, 1983.

[32] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming*. Philadelphia: SIAM, 1994.

[33] X. Ni, M. Verhaegen, A.J. Krijgsman, and H.B. Verbruggen, "A new method for identification and control of nonlinear dynamic systems," *Engineering Applications in Artificial Intelligence*, vol. 9, no. 3, pp. 231–243, 1996.

[34] P.M. Pardalos and M.G.C. Resende, eds., *Handbook of Applied Optimization*. Oxford, UK: Oxford University Press, 2002.

[35] P.M. Pardalos and J.B. Rosen, *Constrained Global Optimization: Algorithms and Applications*. Berlin: Springer Verlag, 1987.

[36] C.L. Phillips and H.T. Nagle, *Digital Control System Analysis and Design*. Englewood Cliffs, New Jersey: Prentice Hall, 3rd ed., 1995.

[37] L.E. Scales, *Introduction to Non-Linear Optimization*. London, UK: MacMillan, 1985.

[38] A. Schrijver, *Theory of Linear and Integer Programming*. Chichester, UK: John Wiley & Sons, 1986.

[39] V. Strejc, *State Space Theory of Discrete Linear Control*. Czechoslovakia: Academia Praha, 1981.

[40] H. te Braake, *Neural Control of Biotechnological Processes*. PhD thesis, Delft University of Technology, Department of Electrical Engineering, Delft, The Netherlands, 1997.

[41] V.J. Terpstra, *Batch Scheduling within the Context of Intelligent Supervisory Process Control*. PhD thesis, Delft University of Technology, Delft, The Netherlands, Oct. 1996.

[42] The MathWorks, *Optimization Toolbox User's Guide – Version 6.0*. Natick, Massachusetts, 2011.

[43] P.P.J. van den Bosch and A.C. van der Klauw, *Modeling, Identification and Simulation of Dynamical Systems*. Boca Raton, Florida: CRC Press, 1994.

[44] S. van Huffel and J. Vandewalle, *The Total Least Squares Problem: Computational Aspects and Analysis*. Philadelphia: SIAM, 1991.

[45] R.J. Vanderbei, *Linear Programming: Foundations and Extensions*, vol. 114 of *International Series in Operations Research and Management Science*. New York, New York: Springer, 3rd ed., 2008.

[46] P. Wolfe, "The simplex method for quadratic programming," *Econometrica*, vol. 27, pp. 382–398, 1959.

Index